Combining and comparing clustering and layout algorithms

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 explores a number of combinations and variants of sampling, K-means clustering and spring models in making such layouts, using Chalmers' 1996 linear iteration time spring model as a benchmark. This algorithm runs in $O(N^2)$ time overall, but the run times for the new algorithms we describe reach $O(N\sqrt{N})$. We compare their layout quality and run times in laying out two collections of synthetic data, drawing samples from each collection of sizes ranging from 1000 to 20000. Based on these comparisons, we outline a number of avenues for future work that may further reduce time complexity and improve layout quality.

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 fundamental 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. Showing these relationships is primary, and manipulation or combination of dimensions is only a means to this end. Our work focuses on creating 2-dimensional representations.

Although effective in generating layouts, standard MDS functions 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 generating 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[5]. 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[4] 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 layout techniques that generate 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

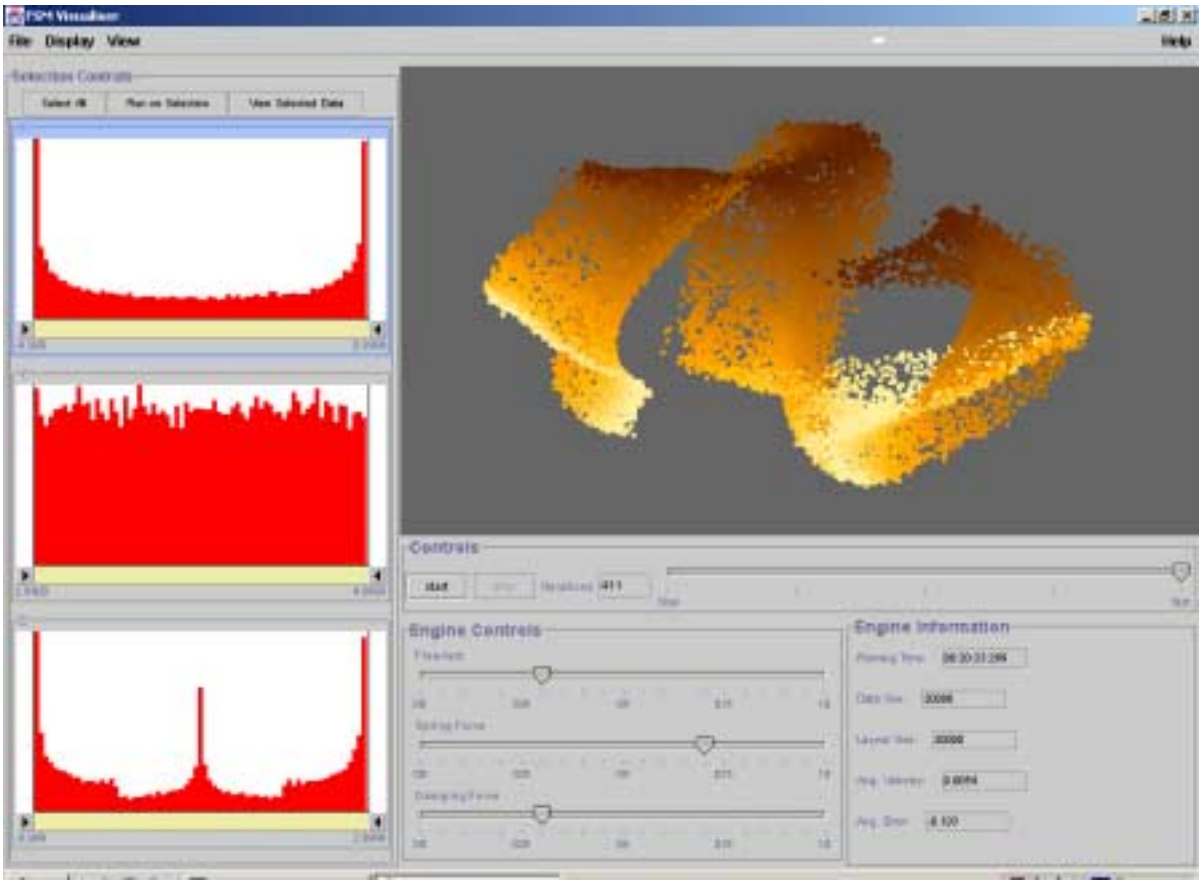

Figure 1. An example layout close to completion in our *FSMvis* visualisation tool is shown top right. Other components of the tool offer control over spring model parameters (bottom right) and histograms (left) of individual dimensions or attributes allow filtering and selection. The layout is of 20000 points sampled from a 3D 'S' shape: one of the test sets described in Section 4. 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.

paper will not focus on the tool in terms of the interaction with the layout, but on new layout algorithms. The following section describes the general approach to iterative layout algorithms that we have been following, namely spring models. A later section outlines the alternatives and refinements we have been working on, and then we report the results of experiments comparing new techniques 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[8] algorithms 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.

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 every pair of objects, $\frac{1}{2}(N^2-N)$ 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-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 problems arise due to the requirement of *N(N-1)* pairwise interactions at each step. This is analogous to the well-known *N-body problem* in computational physics.

2.1 The 96 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 has therefore been reduced from *N(N-1)* to *N(Vmax + Smax)* where *Vmax* is the maximum size of the *V* and *Smax* is the maximum size of *S*. As the two set sizes are bounded by a constant, 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 the best known model using only springs. It is therefore this algorithm that we will use as a basis of comparison with new techniques.

3. Other Techniques and Tools

This section outlines a number of techniques for visualisation and clustering. Apart from older and more established techniques, we add several methods to a selection or toolbox of techniques we later combine and compare.

3.1 Self-organising feature maps

Kohonen's self-organising feature map (SOM)[13] is an unsupervised learning algorithm applied to the classification of information. A representative training set of feature vectors (or patterns) is presented to an artificial neural network. A grid of neurons, sometimes called reference vectors (which are of equal dimensionality to the feature vectors), compete to win the allocation of a feature vector. Local adjustment of reference vectors around each winning neuron gradually produces a topologically ordered structure with each neuron associated with and representative of a cluster within the training set. The grid also provides a useful visualisation tool for inspecting the data set and is often split into *concept areas*[14] to enhance the interpretation of the clusters. While not showing as much topological structure or detail as spring models, SOMs are often quicker to make and scale to larger data sets.

3.2 K-means

K-means, also known as MacQueen's algorithm[15], is a well-known iterative centroid–based divisive clustering algorithm. The algorithm starts by selecting a random sample (of size *k*) from the data set, to approximate the centroids of *k* clusters. Note that we assume here that the input objects consist of vectors of continuous values. Each object in the data set is then compared to each of the centroids, and assigned to a cluster so as to minimise distances in high dimensional space. Once all of the input objects from the data set have been allocated, each centroid is re-computed to reflect the average of its cluster's members. The clustering then is repeated and the centroids are again updated. This iterative process terminates when the algorithm has converged to a minimum and no cluster members change membership.

The main advantage of K-means is that its complexity and time complexity is reasonable[12]. 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[17], it may easily converge to a local minimum. This is compounded by the way that clusters may vary in size and K-means is very sensitive to the initial choice of centroids[12]. This initial decision can determine how well the algorithm converges in terms of local or global minima.

3.3 Hybrid approaches to clustering and layout

Some clustering algorithms effectively tackle the areas in which others are weaker, and a number of researchers have explored combinations of algorithms.

In Su et al[18], K-means was employed to gather representative classes or clusters from the data set. These representative centroids were then organised into a discrete N by N (or more accurately a $\sqrt{k}$ by $\sqrt{k}$) grid. The cells in this array preserve some of the relationships of the clusters, albeit distorted because of the discrete grid structure. The authors hoped to achieve a topologically ordered structure from which the true structure of the data set could be derived. This is exactly in–line with the purpose of Kohonen's SOM, and indeed the SOM is used to fine tune the resulting array if the ordering turns out not to be as good as desired. Su et al suggest that this variant SOM approach is much faster than the traditional on-line SOM.

In a 1998 paper, Brodbeck and Girardin[3] combine a spring model algorithm and a SOM. As their example figures showed, they were able to produce layouts strikingly similar to but far more quickly than a full spring model. Brodbeck and Girardin also describe how they created their own variant of the SOM to handle the classification of categorical (nominal) data through the use of dynamically growing dictionaries.

The SOM finds representative clusters or neurons, and then a spring model is used to layout these neurons without the distortion imposed by the discrete grid of the SOM. Individual members of each cluster, or of only user–selected clusters, can be added to the layout via interpolation. The accuracy of the interpolation is to a large extent determined by the constants in their technique (illustrated in Figure 2):

1. Find the closest neuron within a random subset chosen from all of the neurons.

2. Define a circle round that neuron (in 2D) of radius equal to the high–dimensional distance between the two objects.

3. For a fixed number of random positions on the circumference of the circle, compute the sum of differences between the actual distance and desired (high dimensional) distance of the random position to each of the sampled neurons.

4. Put the cluster member where the sum from step 3 is lowest.

5. Determine the aggregate force vector between this point and the random sample of neurons.

6. Select a constant number of random positions along this vector and, for each, determine the minimum sum of differences (as in step 3) to the sample of neurons. Put the cluster member in this minimal position.

7. Repeat steps 5 and 6 a constant number of times to refine the placement.

As the sizes of all the random samples in the above description can be kept as constants, the interpolation may be achieved with a complexity linear with respect to *N*. The real saving of this method, however, is in the spring model. As only centroids are involved, a far smaller data set is presented to the spring model and, as such, the algorithm will run more quickly. Even though extra interpolation and SOM stages are required, the reduction in input size for the $O(N^2)$ phase results in a saving overall.

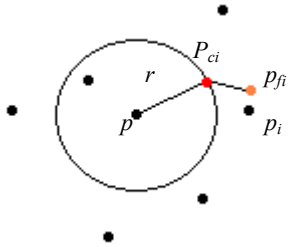

Figure 2. The approximate position of the point p_i, is found by initially finding the optimal position p_{ci} on the circumference of the circle (of radius r from the centroid p). The aggregate force vector is then added to this position p_{fi} to move it closer to p_i.

3.4 Combining Brodbeck & Girardin with K-means

In this section we describe modifications to the Brodbeck & Girardin algorithm. Instead of a SOM, we are exploring K-means as the pre-clustering method. K-means exhibits a lower space and time complexity than the SOM[18]. K-means finds reasonably good

representative points within the data set to facilitate further stages in the layout process.

Based on the Buckshot algorithm[7] we obtain a quick and relatively promising variety of initial centroids. Buckshot works by selecting a random sample of the data set, of size √N, and then applying a clustering routine to that subset, such as group average agglomerative clustering. Once clustering is complete, the centroids are computed and the remainder of the data set is assigned to the closest clusters. In our implementation, we choose $\sqrt{N}$ random samples from the data set and then apply K-means, using this random sample as the initial seed for the clustering. This gives a complexity of $O(lN\sqrt{N})$ for this initial clustering stage, (where l is the number of iterations until K-means convergence).

As before, a spring model is applied to provide a layout of the clusters. We apply the 1996 spring model to the K-means centroids and then interpolate the rest of the data set. A number of criteria are possible for deciding when to stop the spring model. The two main differentials 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 can be applied to any algorithms minimising such a metric, e.g., squared error criterion algorithms such as K-means. Here we define stress as below[4], where d_{ij} denotes the desired, high dimensional distance between objects j and i, and g_{ij} denotes the low–dimensional or layout distance:

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

The interpolation stage in the incremental approach could commence if either the stress or velocity differential falls below a constant scalar threshold. In practice, we based this on velocity.

The interpolation step has also been modified. We use a binary search on the closest quadrant of the circle to find a good position relative to the layout. As before, we find the direction of the aggregate force vector between this point and the layout samples. Then, rather than selecting a position along this vector as before, we add the non–scaled vector to the position. This is because we found that improvements were not always found at Brodbeck & Girardin's position.

3.5 Combining Brodbeck & Girardin with Sampling

To further explore the use of quick approximations, we used a simple $\sqrt{N}$ sample of the data set rather than K-means or a SOM as the initial step. This gives a saving in complexity from the $O(N)$ of the SOM model to $O(\sqrt{N})$. A spring model of this will run in $O(\sqrt{N}.\sqrt{N})$, i.e. $O(N)$. Interpolation follows the same pattern as Brodbeck and Girardin's original $O(N)$ strategy presented in the previous section. However, as no 'parent' information is available, an initial pre-processing stage is required. The remaining $(N-\sqrt{N})$ elements are compared to each of the $\sqrt{N}$ samples. A best fit for all points is consequently calculated in $O(N\sqrt{N})$ time. This pre-processing stage is the dominant factor, making the layout $O(N\sqrt{N})$.

4. Experimental Results

In this section we offer a number of comparisons of three layout algorithms: Chalmers' 1996 algorithm, Brodbeck & Girardin (B&G) with K-means, and B&G with sampling. We evaluate and compare layout algorithms based both on the subjective quality when explored in interactive use on–screen and on objective quantities such as stress.

Since we are trying to minimise this stress value within the shortest period of time, we have also been using a measure of algorithmic efficiency E, combining mechanical stress and time: $E = 1 / ST$. This metric should be used with caution, as stress itself is not necessarily an indication of the quality of the final clustering layout: 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. We also note that in some applications where off-line clustering is permissible, time will not be so important and should therefore not be so prominent in the efficiency metric. Another concern here is that stress is not perceived by users as linear.

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.

Two distinct collections of data were selected for the experiments, each of which synthetically created. The first collection was created by sampling points from a 2D structure—the logo for B&G's company. Reconstructing this structure should be possible for a good layout algorithm. By sampling at different frequencies, sets of 10 different cardinalities were created from this 'logo' set, from 1000 to 10000 elements. The second collection was similarly sampled from a band curving in an 'S' shape through three dimensions. As may be seen from Figure 1, the 'S' structure is forced to fold in on itself in certain areas. It is impossible to exactly represent all the inter-object distances when one less dimension is available. Again, 10 sets were created, this time from 2000 to 20000.

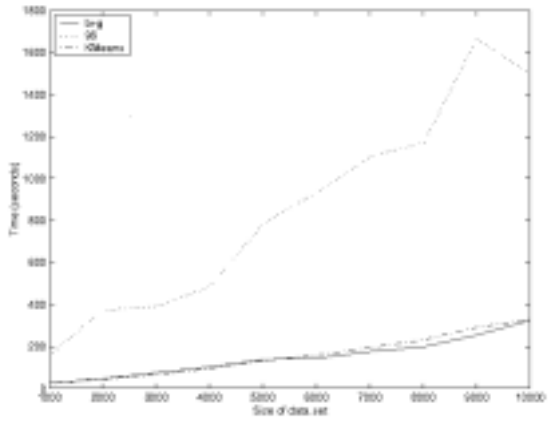

Figure 3. Run time to completion over different sizes of 'logo' data sets

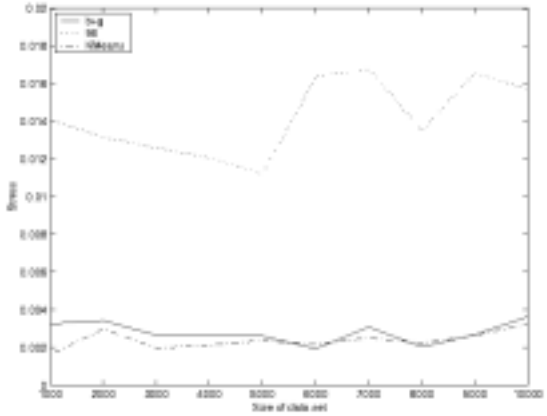

Figure 4. Stress of completed layout over different sizes of 'logo' data sets.

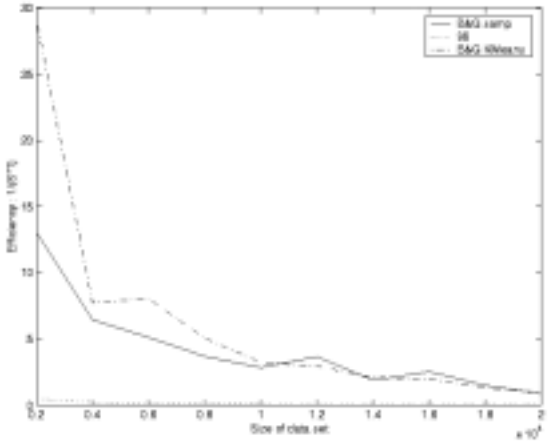

Figure 5. Efficiency for different sizes of 'logo' data sets

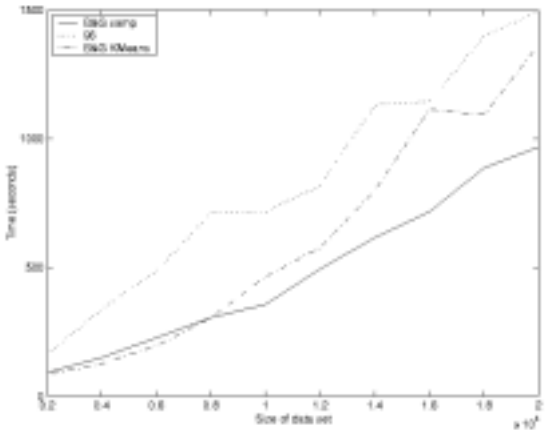

Figure 6. Run time to completion for different sizes of 'S' data.

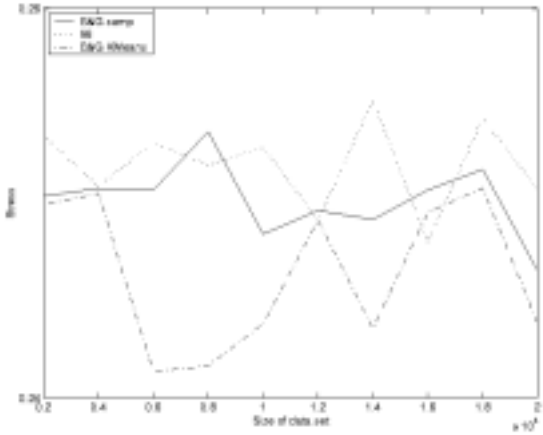

Figure 7. Stress of completed layout over different sizes of 2-dimensional data

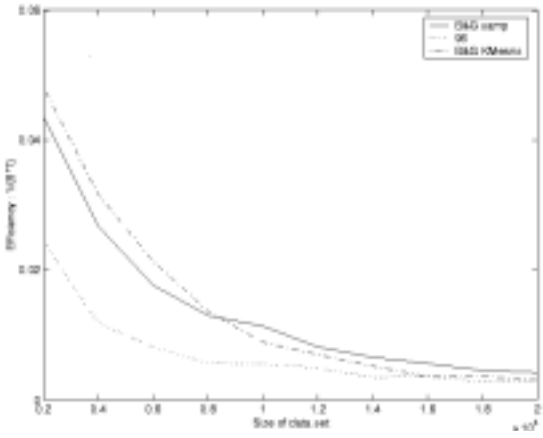

Figure 8. Efficiency over different sizes of 3-dimensional data

Synthetic data were chosen in each case so that we could compare generated layouts and layout processes easily. With the logo data, we could literally see the letters of the logo appearing as the process progressed. Other tests have been performed with real (for example, financial) data, and much of our ongoing work focuses on such data, but the resulting layouts are more difficult to interpret and compare.

Figures 3 and 4 compare of the stress and time to data set size of the '96, sampling B&G and K-means B&G algorithms. As expected, the '96 algorithm takes longer to converge than both the combined algorithms, over all of the data set sizes. We also see that the sampling B&G and K-means B&G exhibit comparable times. For the smaller data sets, K-means appears slightly faster. This is because the K-means algorithm is extremely quick in converging. However, as the size of the data set increases, the cost of using K-means would seem to outweigh its benefit over simple sampling.

The stress recorded for the '96 algorithm is significantly higher than the others. This is because, as the '96 algorithm proceeds, the stress tends to fall sharply and then level out for a considerable time before the layout becomes stable. The stress levels exhibited by the B&G algorithms are lower for the logo set because the interpolation can achieve near-optimal positioning of the points in two dimensions. To enhance this, the 96 algorithm is finally run on this interpolated structure for a small constant number of iterations to refine the layout and minimise the stress.

As Figure 5 shows, both of the interpolation algorithms are deemed by our metric to be more efficient than the '96 algorithm. This graph also seems to illustrate that the K-means version would be the optimal choice for smaller data sets, with this distinction becoming less clear as data size increases.

A similar reduction in computational over the '96 algorithm time may be observed for the second data set (see Figure 6). Indeed, for the set of 20000 data items, over a third of the total running time may be saved through the use of the B&G sampling algorithm.

Although the B&G K-means method takes longer to run than B&G sampling, Figure 7 illustrates that it it does produce lower stress. In general, the stress levels calculated from layouts of the 'S' set are significantly higher than those obtained from the logo set, as would be expected with this inherently difficult layout problem.

From Figure 8, we can again note that K-means appears to be the most efficient method for smaller data sets, while the sampling model becomes the best choice as cardinality increases. Efficiency appears to be much lower in all cases than that calculated from experiments on the first data set. This is again a consequence of constraints on how well 3 dimensional data can be represented in a 2 dimensional display. It should be noted, however, that for data set of 20000 elements, the sampling B&G method was roughly twice as efficient as the '96 algorithm.

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 logo data of size 1000 and 2000 items. The smaller of these data sets was laid out in 1407 seconds (almost 24 minutes) compared to the 21 seconds average over the 10 runs of the B&G K-means model. The data set of 2000 items took 7326 seconds (over two hours) to converge, as compared to 43 seconds average time using B&G K-means. Stress was much higher in the cubic time model (e.g. 0.2) compared to the B&G variants that finished with stresses of roughly 0.001. 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.

5. 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.

5.1 Hashing

In section 3.5, a model was presented where the bottleneck in terms of computational time was 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 each object though the samples. 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 search in high dimensions. Indyk et al. proposed a technique of *locality-sensitive hashing* (LSH)[11] 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[9] 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 can 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 more fundamentally rethink the spring model algorithm.

5.2 Pivots

These algorithms are predominantly used in nearest neighbour searches (NNS) 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[6]: 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.

We believe that this method could be used to improve the '96 algorithm. A pivot-based approach could speed up the operation of building the neighbour sets used in the force calculations.

5.3 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 advisable 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 move towards an overall minimum.

5.4 Batch-SOM algorithm

Earlier in this paper we described our implementation of the K-means algorithm in order to supplant the complexity of the SOM as a pre-clustering step in the modified algorithm of Brodbeck and Girardin. However, Heskes et al[10] describe the implementation of a variant SOM exploiting a batch map principle where the weights in the competitive layer are updated at the end of each algorithmic epoch rather than after each input pattern is presented. It appears that this approach may be superior to K-means but it is faster than the traditional SOM.

5.5 Proximity Grid

In a recent paper[16], 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[18]. It is thought that this discrete layout could be implemented in a way that uses the output of the modified Brodbeck and Girardin 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.

5.6 Interpolation

In modifying the interpolation stage in the B&G algorithm, improvements were to be found in not scaling the aggregate force vector before adding it to the position of the interpolation point. It could be that in some cases the length of this vector should actually be increased, e.g. by applying a binary search along the [0, 2] interval scaled force vector, we may find a better position.

6. Conclusion

This paper has presented preliminary experimentation with a number of combinations and variants of sampling, clustering and spring models. Our data suggests that we can improve upon Chalmers' 1996 algorithm by using the basic approach of Brodbeck and Girardin, with further refinements such as the use of sampling and K-means. These modifications offer run times of $O(N\sqrt{N})$ and layouts of low stress, but further experimentation on data of higher dimensionality and 'real world' complexity will

be necessary before more definitive claims of improvement can be made.

We also devoted a significant proportion of the paper 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 will have to be done before we can say whether such potential can be realised.

7. 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 B&G algorithm.

8. 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. Chalmers, M., "A Linear Iteration Time Layout Algorithm for Visualising High-Dimensional Data", *Proc IEEE Visualization `96*, San Francisco, pp. 127-132 (1996).
5. Chatfield, C., A. J. Collins, *Introduction to Multivariate Analysis*, Chapman & Hall, London (1980).
6. Chávez, E., J. L. Marroquín , G. Navarro, "Fixed Queries Array: A Fast and Economical Data Structure for Proximity Searching", *{\em Multimedia Tools and Applications (MTAP)*, 14(2), pp. 113-135, (2001).
7. Cutting, D. R., D. R. Karger, J. O. Pedersen, J. W. Tukey, "Scatter/Gather: A cluster-based approach to browsing large document collections", *In Proceedings of the 15th Annual International ACM/SIGIR Conference*, pp. 318-329, Copenhagen, (1992).
8. Fruchterman, T., E. Reingold. Graph drawing by force-directed placement. Software—Practice and Experience, 21(11):1129-1164 (1991).
9. Gionis, A., P Indyk, R., Motwani, "Similarity Search in High Dimensions via Hashing", *Proceedings of 25th International Conference on Very Large Data Bases* (1999).
10. Heskes, T., W. Wiegerinck, "A theoretical comparison of batch-mode, on-line, cyclic, and almost cyclic learning", *IEEE Transactions on Neural Networks*, Vol. 7, No. 4 (1996).
11. Indyk, P., R. Motwani, "Approximate nearest neighbours – Towards removing the curse of dimensionality", *Proceedings of SIGMOD '98* (1998).
12. Jain, A. K., M. N. Murty, P. J. Flynn, "Data clustering: A Review", *ACM Computing Surveys*, Vol. 31, No. 3 (1999).
13. Kohonen, T. et al. "Self-organising of a massive document collection", *IEEE Transactions on Neural Networks*, Vol. 11, No. 3 (2000).
14. Lin, X., D. Soergel, G. Marchionini, "A Self-Organising Semantic Map for Information Retrieval", *Proc. ACM SIGIR*, Chicago, pp. 262–269 (1991).
15. MacQueen, J., "Some methods for classification and analysis of multivariate observations", *Proc. 5th Berkeley Symposium on Mathematics and Probability*, pp. 281-297 (1967).
16. 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).
17. Selim, S.Z., M.A. Ismail, "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, (1984).
18. Su, M.-C., H.-T. Chang, "Fast Self-Organizing Feature Map Algorithm", *IEEE Transactions on Neural Networks*, Vol. 11, No. 3, p. 721 (2000).