A Pivot-Based Routine for Improved Parent-Finding in Hybrid MDS

Alistair Morrison
Matthew Chalmers

Corresponding author:
Alistair Morrison
Department of Computing Science,
University of Glasgow
Lilybank Gardens
Glasgow
G12 8QQ
UK

Tel: +44 141 330339
Fax: +44 141 330 4913

morrisaj@dcs.gla.ac.uk

Running title:
Pivots for Parent-Finding in Hybrid MDS

Images
15 images, 3 of which colour: Figures 10, 11 and 12. We request that these all be placed on the same page.

Total number of pages of this document: 48

Abstract

The problem of exploring or visualising data of high dimensionality is central to many tools for information visualisation. Through representing a data set in terms of inter-object proximities, multidimensional scaling may be employed to generate a configuration of objects in low-dimensional space in such a way as to preserve high-dimensional relationships. An algorithm is presented here for a heuristic hybrid model for the generation of such configurations. Building on a model introduced in 2002, the algorithm functions by means of sampling, spring model and interpolation phases.

The most computationally complex stage of the original algorithm involved the execution of a series of nearest-neighbour searches. In this paper, we describe how the complexity of this phase has been reduced by treating all high-dimensional relationships as a set of discretised distances to a constant number of randomly selected items: pivots. In improving this computational bottleneck, the algorithmic complexity is reduced from $O(N\sqrt{N})$ to $O(N^{\frac{5}{4}})$. As well as documenting this improvement, the paper describes evaluation with a data set of 108000 13-dimensional items and a set of 23141 17-dimensional items. Results illustrate that the reduction in complexity is reflected in significantly improved run times and that no negative impact is made upon the quality of layout produced.

A Pivot-Based Routine for Improved Parent-Finding in Hybrid MDS*

Alistair Morrison† Matthew Chalmers‡
Department of Computing Science,
University of Glasgow,
United Kingdom

February 20, 2004

1 Introduction

At the heart of many techniques for information visualisation is the requirement to construct a two-dimensional representation of a multidimensional data set. In arranging data objects in a low-dimensional display, an overview of the data is presented where similar items may be grouped together, allowing insight into general patterns and relationships between individual items or clusters. As these data sets will often have no exact two-dimensional mapping, a configuration of objects is sought that attempts to preserves such high-dimensional relationships. As such, data sets are often treated as *proximity* data and considered in terms of inter-object similarities. Data are positioned so that high-dimensional relationships are represented as well as possible by objects' relative proximities in the created layout. This process of layout creation is known as multidimensional scaling (MDS).

Such low-dimensional configurations are commonplace in many applications that offer overview and insight into large bodies of data. Recently, for example, Andrews et al [2] created the Infosky system for the exploration of a hierarchical

†email:morrisaj@dcs.gla.ac.uk
‡email:matthew@dcs.gla.ac.uk

document collection. Employing a metaphor based on the night sky, documents are represented as stars, with an MDS technique utilised to cluster related items into constellations. Using a similar process, Rodden et al [31] proposed a system for browsing image collections where thumbnail images are arranged in terms of mutual similarity. Experimental evidence suggested that such layouts improved upon random positioning in supporting users' successful retrieval of a desired photo. Another system relying on such techniques [1] supports the exploration of a set of hypothesised evolutionary trees. One component of the system visualises the distribution of trees as a 2D scatterplot, while a second view displays a "consensus tree": a summary of a set of trees that indicates where individual trees agree or differ on ancestory hypotheses. A layout-guiding heuristic was chosen so that a set of neighbouring objects in the resulting 2D scatterplot would produce a well-resolved consensus tree; that is, the individual trees tend to support an agreed hypothesis about evolutionary relationships.

A set of proximity data may be considered as a complete graph, with each object corresponding to a node and each inter-object distance represented as a weighted edge. Eigenvector-based techniques such as the ACE algorithm [19] can be very efficient at positioning nodes in graphs of low connectivity. With a set of multidimensional data, however, we are generally considering relationships to exist between every pair of objects. Cases such as this involving very dense or fully connected graphs are a distinct problem and so we examine alternatives to these matrix methods of node placement.

Spring models (see e.g. [16], [10]) are one possible means of constructing such an object layout. A heuristic algorithm emulating a set of mechanical springs, a spring model updates object positions iteratively in an attempt to structure a 2D configuration in such a way as to preserve high-dimensional distances.

In 2002, an algorithm was presented that combined sampling and spring model phases with a novel interpolation procedure to create representative layouts of multidimensional data in subquadratic time [27] (and extended in [28]). It was shown that the algorithm executed significantly faster than the previous

best spring model algorithm. Figure 1 provides a summary, with the computational complexity of each stage given in square brackets.

It is apparent from Figure 1 that the interpolation stage has the highest complexity, making the model $O(N\sqrt{N})$ overall. Specifically, the parent-finding phase of interpolation is the bottleneck of the model. This paper focuses on this stage of the model. A novel method of parent-finding is introduced that reduces the complexity of the hybrid model to $O(N^{\frac{5}{4}})$. In addition to documenting and evaluating this algorithmic improvement, this paper also provides results of experiments on two data sets; one of 108000 13-dimensional objects, the other of 23141 17-dimensional objects.

The following section provides a summary of related techniques, culminating in a more detailed explanation of the 2002 hybrid algorithm. Thereafter, the purpose of the parent-finding phase of the algorithm is covered in more depth. The original and improved strategies are presented, along with an analysis of their respective computational complexities. An evaluation section follows, which compares the new parent-finding solution to the brute force approach used in the original model in terms of run time, complexity and the accuracy of the parents found. The impact the improved parent-finding routine has on the full algorithm is assessed through a series of comparisons between the original and enhanced hybrid models. The layouts produced by the algorithm are also examined and compared to those generated by neural PCA (principal component analysis [29]); an alternative technique for such problems. Finally, we investigate the role of data dimensionality on the choice of parent-finding routine.

2 Related Techniques

As the presented model is a hybrid algorithm, executing in several stages, it is necessary to introduce some of the techniques from which the algorithm is composed.

To form a layout of N multivariate objects :

1. Select $\sqrt{N}$ subset of objects $[O(\sqrt{N})]$
2. Create 2D layout of subset using Chalmers' [10] linear per iteration spring model $[O(N)]$
3. Interpolate remaining objects onto the layout $[O(N\sqrt{N})]$
   (a) Find parent in sample for each remaining object $[O(N\sqrt{N})]$
   (b) Use high-dimensional distances to a $N^{\frac{1}{4}}$ sample (of the sample) to position remaining objects $[O(N)]$
4. Fine-tune layout with a constant number of iterations of Chalmers' spring model run on the full data set $[O(N)]$

Figure 1: 2002 Algorithm. Complexities are given in square brackets.

2.1 Multidimensional Scaling

The development and use of algorithms for multidimensional scaling may be traced back through several decades. Originating in the psychophysics domain[5], MDS has been a popular tool in the social sciences. Experiments in such fields often gather dissimilarities as the subjective ratings of experimental subjects. In doing so, this permits stimuli (the objects under consideration) to be judged more holistically, rather than on predefined scales. For example, in assessing subjects' views on the relationships between a set of nations, traditional scaling techniques may have necessitated comparisons on set scales (population, economic development etc.). In effect, this is presupposing the criteria by which a subject will judge the data. With an MDS model, an experimental participant could be asked merely to state the perceived similarity between each pair of nations. MDS can then be used to produce a layout configuration of the nations and analysis thereupon may determine patterns or structure that could indicate 'cognitive dimensions' behind subjects' judgements. As such, MDS allows the data themselves to determine subjects' judgement criteria (and even the

dimensionality - the number of such criteria) without the need for an a priori theoretical model.

The MDS problem has been defined in several distinct ways, and algorithmic solutions can take many forms. It is beyond the scope of this paper to survey the entire field, but interested readers are directed towards the following landmark papers [35][34][20][9], or [21][5] for a general overview. In this paper, the focus will be on several heuristic models for the creation of 2D configurations of higher-dimensional data sets, as described in the following sections.

2.2 Distance metric

As stated, MDS algorithms take as input a set of proximity data: a matrix of similarities (or, more often, dissimilarities) between a set of objects. Given a set of high-dimensional data, therefore, it is necessary to compute the dissimilarity between each pair of objects. In the following discussion, we consider a data set to be represented by an N x d rectangular matrix. Each row is therefore a d-dimensional object.

We use a simple metric of Euclidean distance to calculate high-dimensional dissimilarities. Eq. (1) illustrates how dissimilarities are computed between objects a and b, where we use the notation x_i to denote the value in column i of object x.

$$Dissimilarity(a,b) = \sqrt{\sum_{1 \leq i \leq d} (a_i - b_i)^2} \tag{1}$$

2.3 Spring Models

Of the various methods for the layout of high-dimensional data sets, the spring model [16] [15] [10] is among the simplest. Originally proposed by Eades as a heuristic for graph drawing [14], a spring model moves to equilibrium a system of steel rings connected by springs. In Eades' original algorithm, springs were used to simulate graph edges. Objects were initially positioned at random, with springs therefore being arbitrarily contracted or stretched. The forces generated

by the springs were then used to iteratively pull or push the objects to a position of equilibrium and an aesthetically pleasing graph layout resulted.

One of Eades' criteria for this notion of aesthetics was the uniformity of edge (spring) lengths. In the visualisation of graphs, it is generally considered desirable to represent edges as being of equal length. In contrast, MDS seeks to use inter-object spacing on a layout to represent high-dimensional similarities. In treating multidimensional data, therefore, the ideal (relaxed) length of each spring between two objects is set to be proportional to the high-dimensional dissimilarity between the objects.

Forces in the system are computed as being proportional to the difference between the high-dimensional (ideal) distance and the low-dimensional (current) layout distance, and the system strives to minimise the difference between these two. A loss function can be derived, known as stress, that 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 the metric in Eq. (2) (commonly known as Stress-1 [20]), where h_{ij} denotes the desired, high dimensional distance between objects i and j, and l_{ij} denotes the low-dimensional or layout distance:

$$Stress = \frac{\sum_{i<j}(h_{ij} - l_{ij})^2}{\sum_{i<j} l_{ij}^2} \tag{2}$$

As previously remarked, a set of multidimensional data may be considered as a fully connected graph. Consequently, a spring is required between every pair of objects. In calculating the force acting upon an object, it is therefore necessary to calculate the aggregate force of $(N-1)$ springs. It follows that to move all the objects, $\frac{1}{2}N(N-1)$ operations are necessary, and, since the number of iterations required to reach equilibrium is commonly proportional to N, the model has computational complexity of $O(N^3)$. The $N(N-1)$ pairwise interactions at each step are an obvious area for improvement.

Chalmers' 1996 spring algorithm [10] helped alleviate this burden through

a system of caching and stochastic sampling. A constant-sized set of "neighbours" was maintained for each object i; those items most similar to i in high-dimensional space. Through limiting force calculations performed for object i to the members of this set, and a constant number of random objects, the algorithm enabled each iteration to be performed in linear time, thus permitting the construction of a stable layout in $O(N^2)$ time overall. In terms of computational time, this is currently the best model using only springs. It should be noted however that such a model could not be practically used on data sets over a few thousand objects in size.

[28] provides a more detailed description of the mechanics of spring models and their use within our hybrid framework.

2.4 The 2002 Hybrid Layout Algorithm

Many models for clustering or layout are specialised to certain types or characteristics of data. It is therefore commonplace that certain algorithms will be effective when applied to one type of task, but weaker on others. We have found it beneficial to create hybrid, or combinatorial, techniques that maximise the benefits of different approaches, while diminishing the impact of any shortcomings. Figure 1 provides an outline of our 2002 algorithm, along with the computational complexity of each phase.

In the first step, a $\sqrt{N}$ subset is sampled from the data (where N represents the number of objects in the set). A 2-dimensional layout of this $\sqrt{N}$ subset is then generated via Chalmers' spring model algorithm [10]. It is hoped that a $\sqrt{N}$ subset will be large enough to provide a representative overview of the data, yet small enough to permit the $O(N^2)$ spring model phase to be completed in linear time.

It is interesting to note that, due to the multi-stage nature of the algorithm, it is possible to halt execution at this stage. An overview of the data set has been created in time proportional to $O(N)$ and a user has the option of proceeding with the full layout, or abandoning execution. From an interactive point of

view, this could be of benefit in allowing a user to rapidly generate overviews of a number of different data sets and then proceeding to examine the most interesting sets in detail, without having to wait for full layout generation of the less interesting sets. Such interaction is explicitly supported in the implementation in the HIVE framework [33], as described in section 4.1.

Stage 3 of the model uses the generated subset layout of stage 2 to calculate locations for each of the remaining $N - \sqrt{N}$ objects. This procedure comprises two steps. Every object i in the $N - \sqrt{N}$ interpolation set is first assigned to its "parent" x in the subset; the object in the $\sqrt{N}$ layout most similar to i. A procedure follows, illustrated in Figure 2, that positions each object i ideally with respect to an $N^{\frac{1}{4}}$ subset of the objects. To begin, a circle is defined around the parent object with radius proportional to the high-dimensional distance between the parent and i. A binary search is then performed on the circumference of this circle to determine the object's ideal location. The object will therefore be placed ideally with regard to its parent, while still taking into account relationships with other sample objects.

A potential location l for object i is evaluated through summing discrepancies between high and low-dimensional distances to the subset objects. The distance is measured from l to the layout position of each subset object z and compared to the high-dimensional dissimilarity between i and z. That is, the following is calculated for several potential locations, with the object being positioned at that location giving the lowest result.

$$\sum_{1<z<N^{\frac{1}{4}}} |layoutDist(position_z, l) - highDDist(z, i)| \tag{3}$$

Finally, a constant number of force vectors are calculated, based on summed distances to the $N^{\frac{1}{4}}$ subset. These forces are used to iteratively update the object's position in a manner similar to spring iterations.

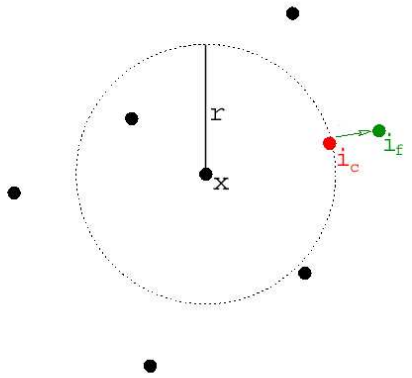


Figure 2: The procedure to place object i onto the sample layout. The object's 'parent' x is determined, and a circle is defined, centred on x. The radius r is equal to the high-dimensional distance between x and i. Object i's position is then determined via a minimisation of Eq. (3); the summed discrepancy between high and low dimensional distances. An ideal location on the circumference of the circle, i_c, is determined via a binary search, then a series of composite force vectors are added to give the final location i_f.

3 Reducing Complexity With Pivots

This paper describes a faster means of achieving the first stage of interpolation (step 3(a) in Figure 1). The spring model run on the original $\sqrt{N}$ sample has completed, and the remaining $N - \sqrt{N}$ objects in the data set must be mapped onto this layout. The first stage of this process is the assignment of each remaining object to a "parent" in the sample layout. The interpolation of an object begins with the creation of a circle around its parent, with radius proportional to the high-dimensional distance between object and parent. From the description of the technique in the previous section and [28] it is clear that the accuracy of placement will to a large extent be governed by the size of this circle.

The similarity in high-dimensional space between an object and its parent will therefore determine how close the object is placed to its ideal location.

3.1 Parent Selection: A Near-Neighbour Search

This parent-finding stage is an example of near-neighbour search. A near-neighbour from the $\sqrt{N}$ sample layout is sought for every point to be placed. We desire the best possible approximation to the point's closest neighbour in order to maximise the accuracy of the point's placement.

Near-neighbour searching is considered a fundamental task in computing science, and has applications in fields as diverse as information retrieval, data compression and pattern recognition [17]; any area, in fact, where it is possible to treat data as a "metric space" [12]. In such a space, a distance function may be defined, with which inter-object dissimilarities may be assessed. For our purposes, we look at the more specific case of a "vector space", in which objects are defined as tuples of real numbers (i.e. a high-dimensional data set).

The central idea in most algorithms for efficient near-neighbour search is to preprocess the data set in such a manner as to facilitate ease of search, limiting the number of distance calculations required. Chávez et al [12] survey a number of algorithms for the general metric space and organise them into a taxonomy under a common unifying model. Two main classes emerge: clustering techniques and pivot-based algorithms. Models in the former of these classes [7][13] divide data spatially into several compact zones, often based on Voronoi-type partitioning (see e.g. [3]). Zones, and the objects therein, may then be discarded from consideration in near-neighbour queries through an efficient number of distance calculations. The second class; pivot-based algorithms [8], [36]; select a certain number of objects within the data sets to act as "pivots". Rather than dissecting the space geometrically, objects are classified by their proximities to the pivots. Distances from a query object may then be used in conjunction with the triangular inequality (a fundamental property of distance functions in metric space, see [12]) to discard elements from consideration. An alternative method functions via the creation of a metric embedding of the data objects in a lower dimensional space. Searches may then be undertaken in this embedding space. Lipschitz embeddings[6] provide guaranteed bounds on distortions and

have been shown to provide effective results, although this approach is generally applied to data sets of far higher dimensionality than the experimental sets presented in this paper. As will be shown in section 3.1.3, our chosen model for parent-finding is most akin to the pivot-based searches.

Although the problem of near-neighbour searching was first studied in the 1960s [24] and research into this area continued in the following years, there has been little improvement in efficiency, especially when dealing with sets of high dimensionality [18]. This trend, often termed "the curse of dimensionality", refers to the property of nearest-neighbour algorithms often having exponential dependencies on space and run times with regard to the number of dimensions.

There is a strong belief that the "intrinsic dimensionality" of a space may in fact be the property impacting on complexity. The intrinsic dimensionality may be imagined as the number of dimensions required to represent a data set without losing distance information. This could explain the common observation that some algorithms seem to perform unexpectedly well given worst case analyses [23]; the complexity-determining dimensionality of a data set is significantly reduced through efficient preprocessing. A formal definition is provided in [12], where it is claimed that no technique can reasonably cope with an intrinsic dimensionality above 20.

3.1.1 Brute force approach

In the original hybrid algorithm, a brute force approach was adopted in finding parents whereby a linear search was executed on the subset of objects making up the original sample layout. A distance calculation was performed between the object to be interpolated and every item in the sample layout, with the item yielding the least distance chosen as the parent.

Pseudocode for this brute force approach can be written as follows.

For all $N - \sqrt{N}$ yet to be laid out

    For all $\sqrt{N}$ in sample

        Perform distance calculation

The resultant complexity can be calculated as

$(N - \sqrt{N})\sqrt{N}D = N\sqrt{N}D - ND = O(N^{\frac{3}{2}}D)$

(where D represents a high-dimensional distance calculation).

3.1.2 Random sampling

A saving in computation may be achieved by selecting a sample of the original subset on which to base parent searches. A linear search is still required, but this search executes on a far smaller set of objects than the previous method. It is hoped that a representative sample may be selected, so that the quality of parent found will not be greatly affected by this shortcut. Assuming a sub-subset of $\sqrt{samplesize}$ (size $N^{\frac{1}{4}}$), this would execute as:

For all $N - \sqrt{N}$ yet to be laid out

For all $N^{\frac{1}{4}}$ in sample of sample

Perform distance calculation

$(N - \sqrt{N})N^{\frac{1}{4}}D = N^{\frac{5}{4}}D - N^{\frac{3}{4}}D = O(N^{\frac{5}{4}}D)$

This demonstrates a significant saving over the previous brute-force method.

It should be noted that, although quicker, this approach will not always select the best possible object to act as the parent. Consequently, object placement during interpolation will be less accurate than that achieved through use of the full brute force approach. It is for this reason that the more computationally complex brute force model was implemented in the original hybrid model.

3.1.3 Pivot-based parent-finding

This section describes a novel routine whereby the complexity of parent-finding is reduced without impacting on the quality of parent selected.

This near-neighbour search algorithm is based upon the pivot-based method of dimensional reduction. First used in Burkhard-Keller Trees [8] as a means of hierarchical binary decomposition of a vector space, pivots are now being used

as the basis for techniques such as the Fixed Queries Array [11] for proximity searching.

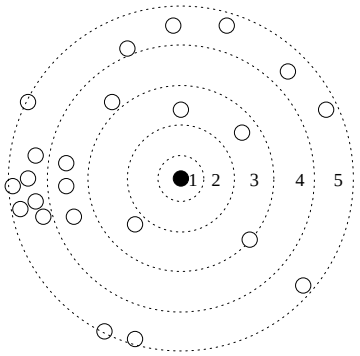


Figure 3: Diagram of one pivot object (represented by the shaded point). A pivot has a certain amount of buckets, shown as numbered discs between the dotted circles. Each remaining data item is stored in one bucket, as determined by its proximity to the pivot.

As mentioned, most efficient near-neighbour search algorithms function on the premise that preprocessing a data set can reduce the work necessary at query time, and hopefully reduce the number of operations required overall. To preprocess, we randomly select k points from within the data set to act as "pivots". Pivots are treated as having a certain number of buckets, each representing different ranges of distance from the pivot (as shown in Figure 3). The rest of the data set may be stored in these buckets as determined by proximity to the pivot. In doing so, the data dimensionality may be reduced to k through the representation of each point only as a set of discretised distances from the pivots.

For our purposes, we appoint a certain number of objects from the $\sqrt{N}$ sample (selected in step 1 of Figure 1) to act as pivots. Preprocessing involves assigning each non-pivot in this $\sqrt{N}$ sample to a bucket for each of k pivots.

Thereafter, when we wish to find an object's parent, a distance calculation is performed between the object and each of the pivots. From this distance calculation, the appropriate bucket for each pivot is determined, and the contents of each of these buckets are searched for the overall nearest neighbour.

In these calculations, we assume a constant number of pivots, k, and that the number of buckets chosen for each pivot is $\sqrt{samplesize} = N^{\frac{1}{4}}$.

Preprocessing Build data structure:

For all $\sqrt{N}$ in sample

For constant number of pivots

Perform distance calculation

Query Find parent for object q:

For p in 1..constant number of pivots

Distance calculation // *determine bucket for* q *in* p

// *Find closest point in this bucket*

For i in 1..number of points in bucket

Perform distance calculation

The complexity of the preprocessing stage is simply $\sqrt{N}kD = O(\sqrt{N}D)$.

When performing a query, we have the following average case complexity (the average number of points in a bucket is represented by $\frac{sampleSize}{numBuckets}$).
$k\frac{\sqrt{N}}{N^{\frac{1}{4}}}D = O(N^{\frac{1}{4}}D)$

The query will be performed for all $N - \sqrt{N}$ points not yet placed, so will be
$(N - \sqrt{N})N^{\frac{1}{4}}D = O(N^{\frac{5}{4}}D)$

Overall, then, complexity will be
$O(\sqrt{N}D) + O(N^{\frac{5}{4}}D) = O(N^{\frac{5}{4}}D)$

Again, this is a considerable saving in complexity over brute-force; equivalent in fact to the previously described sampling method.

It is worth emphasising that this analysis is based upon average-case performance. A worst-case situation would arise if all the objects were to fall into the

same bucket for all pivots. This situation could conceivably arise if a data set consisted of a very tight cluster and a number of remote outlier objects, with the pivots being chosen from the outliers. This is very unlikely as we would expect the sampling used in pivot selection to reflect object distribution. In this case, however, the entire subset would have to be searched and the complexity would therefore return to $O(N^{\frac{3}{2}}D)$.

It can be seen, then, that the worst-case complexity of this method is as good as the previously used brute-force approach. Moreover, this worst-case is a remote possibility and we expect significantly better performance in the grand majority of instances.

4 Experimental comparison of parent-finding methods

In this section, the three methods of parent-finding outlined in section 3.1 are evaluated experimentally: brute force, random sampling, and pivot-based. Execution times for each method are graphed, as are measures of layout quality as determined via the stress metric given in Eq. (2). The impact of the choice of parent-finding method on the hybrid model is also explored by comparing run times with the non-pivot-enhanced model, and a comparison is made with a layout derived from neural PCA; an alternative layout technique.

Evaluation took the form of a series of test runs of each algorithm on two data sets. The first, a set of audio data, were sampled from British television broadcasts during the 2002 FIFA World Cup as part of an investigation into the application of audio-based event detection to sporting events [4]. 108000 seconds of audio were recorded, with each second treated as an object to be visualised.

A completed layout of the audio data is shown later in Figure 10. Two main clusters are apparent in the data: AB and C. AB has two subclusters, labelled A and B in the figure. Through isolating individual objects and listening to

the associated audio clips, we can deduce that the left-most of the two visible structures represents speech, with the section labelled A corresponding to in-match commentary and section B comprising studio-based pre or post-match analysis. Cluster C represents music occurring during the broadcasts.

Discussions were conducted with the domain experts as to how the audio data should be processed for use in MDS experiments. It was decided that the experiment set should be generated using Linear Predictive Coding (see [30] for an introduction) to create a 13-dimensional data set, with each dimension representing a weighted cepstral coefficient.

Our second data set was gathered during an investigation into carbon cycling in Antarctic lakes, undertaken as a collaborative eScience project between the Australian Antarctic Division and the Equator Interdisciplinary Research Collaboration. Data were gathered at 5 minute intervals by a remote sensing probe between the dates of the 12th of May and the 31st of July 2003. Each object represents a 'snapshot' of data from the probe at a particular moment. Measurements taken include air and water temperatures and photosynthetically active radiation at various depths. The data set was 23141 objects of 17 dimensions. A completed layout of this data is shown as the image on the left of Figure 11. More information on the data and project is available at http://www.equator.ac.uk.

4.1 Implementation

The following results are taken from a series of experiments performed on a PC with a Pentium 4 2.41 GHz processor, 504 MB RAM and Windows XP Professional 2002.

All code was written in Java SDK 1.4.1. The algorithms were implemented in HIVE (Hybrid Information Visualisation Environment), a software system that supports the construction of hybrid algorithms through the representation of algorithmic components as modules in a visual programming environment [33]. Figure 4 illustrates our algorithm. Links between modules simulate data flow,

with the results of each operation passed to the following component. To measure performance, modules were added to facilitate algorithmic profiling. Tools to measure time and stress may be applied to individual algorithmic stages, or the overall model. The Multiple Runs module may be given a command-line style argument to control a batch job of algorithmic executions. In this way, parameters may also be specified for each module, providing, for example, instructions on which parent-finding routine to perform, or the number of pivots to select.

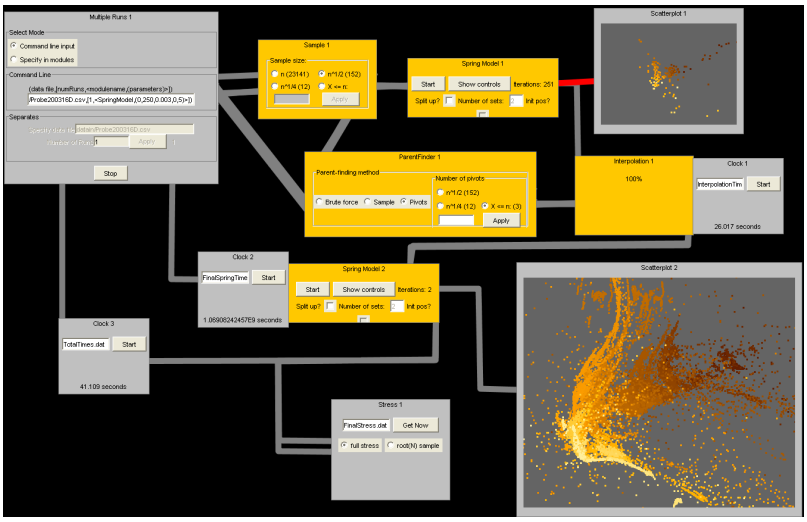


Figure 4: An implementation of the algorithm in the HIVE framework. The Multiple Runs module (top left) controls a batch job of executions under different conditions and with different data sets. Clock and Stress modules measure performance and dump results to files. The algorithm is running on the 17-dimensional Antarctic data.

4.2 Run times for parent-finding

It has been shown that both random sampling and pivot-based selection are of lower computational complexity than the brute force approach. Figure 5 further illustrates that the distinction in complexity is reflected in run times.

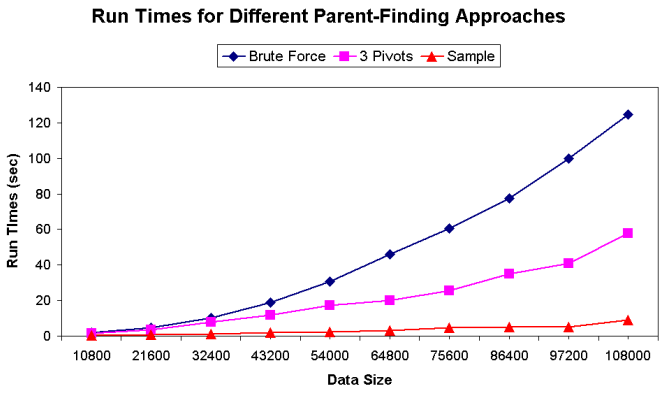


Figure 5: Time taken to complete each of three methods of parent-finding on 13-dimensional data sets of sizes 10800-108000 objects. The graph displays mean results of six runs performed on each size using each model.

The audio data has been sampled to generate ten sets ranging in size between 10800 and 108000 objects. The graph shows results averaged over six runs of each model using each size. A choice existed here for the constant number of pivots to use, as discussed in the following section. For these experiments, we opted to use 3 pivots.

As predicted from complexity calculations, the brute force method is the most time-consuming for all data sizes. It is also apparent that the sampling method is the quicker of the two low-complexity models. However, as the following sections illustrate, this saving in time comes at the expense of accuracy

of results.

4.3 Accuracy of parent found

A simple test was conducted to determine the effectiveness of the parent-finding algorithms. High-dimensional distances were calculated between every two objects in a data set then ordered such that, for every object, a list was constructed ordering every other object in terms of proximity. That is, for element x, the first item in the list would be x's nearest neighbour in the full set, the second item the second closest to x and so on.

Once this list had been created, a parent-finding algorithm was run for the $N - \sqrt{N}$ objects to be interpolated. For each of these objects, the quality of the parent found was assessed by its proximity to the head of the list of nearest neighbours. The results were averaged over all $N - \sqrt{N}$ searches.

The results, taken from a set of 1000 items and averaged over 10 runs for each method, are shown in the table below. The brute force and random sampling conditions were tested, as was the pivot-based search, with different values selected for the constant number of pivots. A further case is shown whereby a completely random member of the subset is chosen in each case to be the parent.

Method	Rank
Brute Force	30
Sample	110
1 pivot	59
2 pivots	37
3 pivots	35
4 pivots	34
10 pivots	32
Random	486

Table 1: Accuracy of parents found, as determined by rank in list of nearest neighbours.

We can see from the table that the sampling method may have been the quickest in Figure 5, but it has produced significantly less accurate parents than

its two competitor techniques. Conversely, although the pivot-based method of parent-finding has considerably lower complexity than the brute force approach, the quality of parent found is comparable. The forthcoming sections discuss the trade-off between accuracy and run time for the parent-finding stage.

Obviously, for any given interpolation object, it is unlikely that the ideal neighbour would be in the $\sqrt{N}$ sample. As the brute force model is guaranteed to find the best possible neighbour from the sample, we see that the best possible results we can hope for are roughly the $\sqrt{N}$'th best neighbour.

The number of pivots used in the algorithm would appear to be of importance, with the results suggesting that the more pivots used, the closer the performance approximates the brute force method.

4.3.1 Cluster centroids as parents

As a point of interest, if the interpolation phase is based on a layout arising from k-means clustering [22],[26] rather than random sampling, we can expect to do rather better than finding the $\sqrt{N}$'th best neighbour. Consider Figure 6, where the sample layout of cluster centroids is uniformly spaced. In terms of distance from a parent, the worst case we could imagine is a point on the boundary of two cluster regions (point A). If we assume that the data are evenly distributed (this will obviously not be the case in an average data set, but serves to illustrate this example), with $\sqrt{N}$ points in each of the $\sqrt{N}$ clusters, we would expect a point such as A to be the furthest point from that parent. Hence, the parent for point A would be the $\sqrt{N}$'th nearest point to A. Similarly, a point such as B would be the nearest neighbour to its parent.

On average (again under conditions of even distribution), one would expect a parent to be the $\frac{1}{2}\sqrt{N}$'th nearest neighbour to a query point. This is indeed what was discovered, as the brute force method applied to a layout of k-means centroids yields an average result of 16th nearest neighbour for a 1000 element data set.

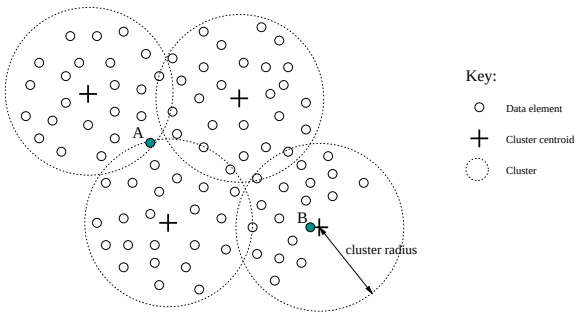


Figure 6: 2-dimensional layout of an approximately evenly-distributed data set with imposed clustering. Point A illustrates a worst-case example of distance from the parent, point B a best case.

4.4 Post-interpolation stress

We have outlined three methods of parent-finding and their effectiveness at selecting a near-neighbour. It is now necessary to assess the impact of choice of parent on layout quality. The quality of a layout is calculated via the stress-1 metric outlined in Eq. (2). A lower stress value indicates a better layout.

For each of the parent-finding methods, we calculate the stress after the completion of the interpolation phase, again averaged over six runs of each size of the audio data. As before, 3 pivots were used during the experiments.

As may be seen from Figure 7, stress levels may be somewhat erratic. This is due to the interpolation being very dependent on the initial random sampling and spring model phases. Despite the fluctuations, we can see that both the brute force and pivot-based methods exhibit lower stress than the sampling approach. As explained earlier, the brute force method will provide the lowest stress that we could expect for any given run, so it is a side-effect of sampling that we see the pivot-based method lower on some sizes. This does, however, serve to illustrate that the two methods yield similar stress levels, and continue to do so as data size increases.

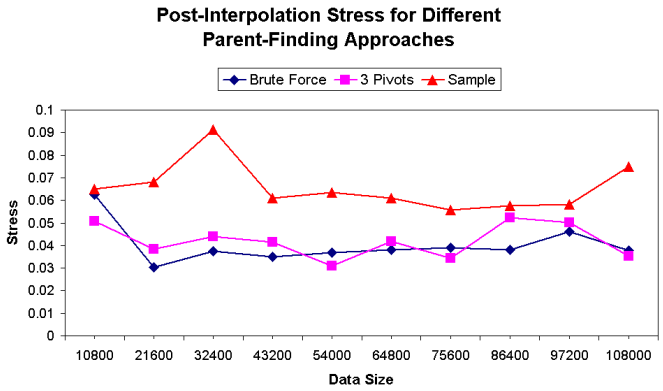


Figure 7: Post-interpolation stress levels across different data sizes with 3 different parent-finders.

4.5 Effect on full hybrid model

It has been established that the pivot-based method of parent-finding reduces complexity and run times from the brute force approach, while still finding accurate neighbours and therefore creating layouts of low stress. In this section, an investigation is conducted into the effect of the choice of parent-finding model on the full hybrid model.

The stress present in the layout after interpolation was given in section 4.4. We aim to reduce this stress in the final stage of our visualisation technique, where a linear per iteration spring model is run over the entire data set (step 4 in Figure 1). To attempt to find an optimal layout, the spring model may be set to terminate when the average velocity drops below a specified threshold; an effect we expect when the layout has converged to a state of low stress.

We would predict that a layout with a higher stress level after interpolation

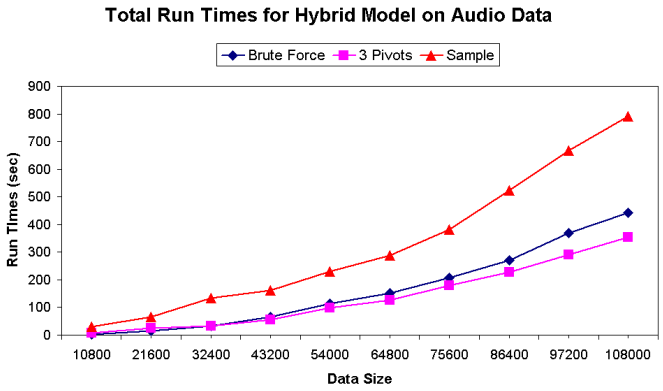


Figure 8: Total times for layout generation with each of the parent-finding methods executing on the 13D audio data.

would require more iterations of this final spring model until termination and therefore have longer run times overall.

Figure 8 (detailing experiments averaged over six runs on the same audio data) illustrates the total times for the complete visualisation process. Here we see that the pivot-enhanced model is clearly the fastest of the three. The model using random sampling was, as expected, by far the slowest, due to the extra iterations of the spring model required to lower its high stress. A similar pattern may be observed when the models are run on the Antarctic data in Figure 9.

From this, it can be concluded that it is worth investing the extra effort in the parent-finding phase. Although techniques such as brute force and pivots take more time at this stage, the interpolation is performed more accurately, and, as such, the required number of final iterations is reduced, resulting in a saving of time overall.

The pivot-based model is the quickest overall due to the low complexity and

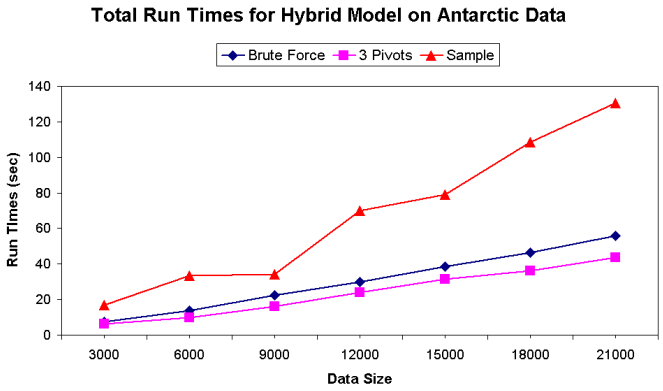


Figure 9: Total times for layout generation of the 17D Antarctic data set.

accuracy of its parent-finding approach.

As the model has been set to terminate automatically when the layout approaches stability, layout qualities are expected to be constant across all variants of parent-finding procedure. This is indeed the case, as illustrated in table 2, which shows final stress values for each of the 3 methods executing on each data set.

Method	Audio data	Antarctic data
Brute Force	0.02831	0.2512
Sample	0.02849	0.2481
3 pivots	0.02834	0.2562

Table 2: Final stress values for each parent-finding method, on both data sets. Stress values, as expected, are very similar at the end of execution, as our algorithm terminates automatically when layout stability is achieved.

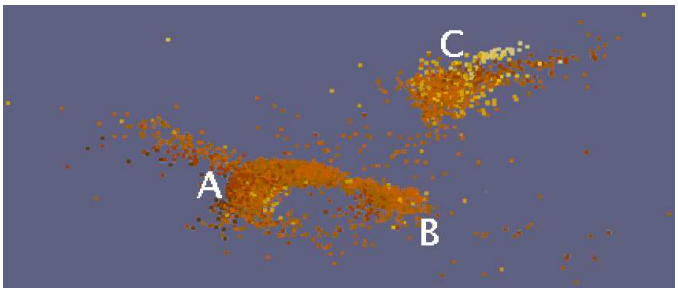


Figure 10: Completed layout of audio data using hybrid MDS algorithm. Each point represents one second of sound. The clusters labelled A and B correspond to speech, while C represents music.

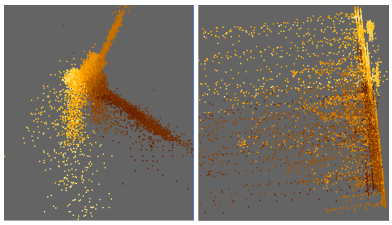


Figure 11: Layouts of 17D Antarctic data using hybrid algorithm (left) and neural PCA (right).

4.6 Comparison with An Alternative Layout Technique

In the previous sections we have shown that the algorithm performs well in comparison with its predecessor hybrid model. In order to broaden evaluation, we also offer a comparison with neural PCA[29], an alternative technique for the layout of high-dimensional data.

Figure 11 represents the configurations of the Antarctic data generated by each of the two techniques. The plot on the left was generated by our hybrid model and that on the right was the output from neural PCA. Both plots are coloured by time - one of the original input dimensions. It can be seen that the two scatterplots are markedly different. The hybrid model appears to have arranged objects together if they were captured at a similar time. The neural

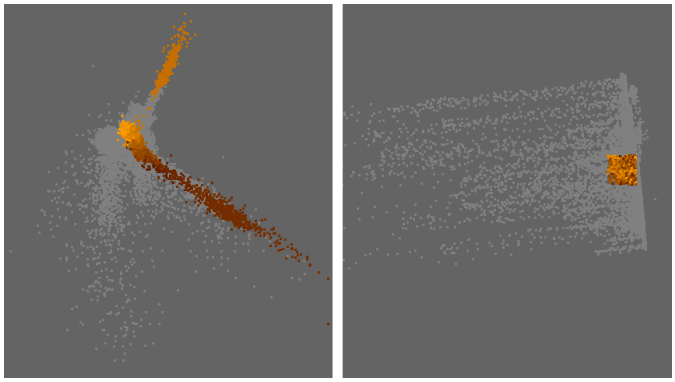


Figure 12: Selecting objects in the PCA plot (right) highlights the corresponding objects in the hybrid algorithm output. The hybrid algorithm has identified two separate groups at the extremities of the layout. PCA has positioned the same objects in a dense region, not distinguishing separate clusters.

PCA output makes less of a distinction between the different periods, and has most of the data squeezed into a long band running down the far right of the layout. The clustering detected by the hybrid model would appear to be representative of the data, as objects from the same time period will tend to share similar values in other dimensions.

Figure 12 illustrates further. HIVE supports brushing between multiple views, whereby the selecting of objects in one view will highlight the corresponding objects in another. The figure illustrates the case where a small rectangle has been highlighted in the PCA plot, selecting a densely packed region of objects. It can be seen that the same data have been treated very differently by the hybrid model. The objects have been spread out in two (almost orthogonal) directions, and are treated as extremities of the layout. The hybrid layout also suggests that there are two separate clusters within the selected subset. The

colouring in the topmost of these clusters indicates that the objects represented therein were captured over a relatively short time period. Further examination indicates that these objects also share very similar values in the ice thickness and air temperature dimensions. This partitioning is not represented in the PCA output, as the layout is structured so as to better represent the higher variance in other dimensions. Similar results were noted in [32], where a layout from the hybrid model of [27] managed to separate an outlying cluster that the neural PCA method could not detect.

In an exploratory analysis of the data, it would obviously be of benefit to examine the output from both of these methods. The purpose of this section is not to argue a case for either of the layouts being of greater value, but to illustrate that there is room for both techniques in the exploratory data analyst's arsenal.

4.7 Parent-Finding and Dimensionality

The purpose of the parent-finding algorithm is to reduce the number of distance calculations performed. It is important to distinguish this from the total number of operations required overall. Although an efficient data structure may limit the number of comparisons necessary to select a representative parent object, the additional overhead demands of such a technique can prove prohibitive. The space complexity of nearest neighbour searching techniques is always an important consideration. It has been noted [12] that pivot-based techniques are theoretically the most computationally efficient, should the algorithms' memory requirements be met. These requirements are often practically infeasible, although techniques such as FQA [11] allow the user to select the number of pivots used and therefore effectively tradeoff speed for memory.

In our parent-finding task, whether such a preprocessing scheme is worthwhile is dependant on the dimensionality of the data set. To illustrate, imagine a data set of 1000 200-dimensional tuples. The Euclidean distance between 2 objects of such a set would be calculated as in Eq. (1), defined earlier. Every

distance calculation performed during parent-finding would require 200 subtractions and squaring operations.

The pivot procedure's ability to substantially reduce the number of distance comparisons is of great benefit under such conditions, and the few thousand additional operations required to prepare the data structure would be of little comparative hindrance.

With a 3-dimensional data set, however, the same amount of preparatory operations are necessary, but each distance calculation avoided would only save 3 operations. Thus, the pivot procedure's initialisation would constitute a larger percentage of the run time, and would perhaps even be in excess of the savings gained through limiting distance calculations.

An experiment was performed to explore this issue and determine the dimensionality at which the pivot-based method becomes the algorithm of choice. An environmental scenario data set was used in these tests, courtesy of the University of British Columbia's Sustainable Development Research Initiative, Georgia Basin Futures Project, and Envision Sustainability Tools. The data were composed of 5000 objects of 293-dimensions. Random sampling was used as in the previous experiments to generate subsets of the data on which to run the experiment. Rather than sampling the data cardinality, however, random dimensions were selected to generate sets with dimensionality ranging between 2 and 280. Such a process would obviously lead to meaningless layouts, as we are arbitrarily discarding information, with no consideration as to dimensions' importance in any sense. However, we feel that this process is justified, as this is a purely arithmetic exercise and we do not intend to give any consideration to the generated layouts.

Figure 13 shows results of these tests. It may be seen that the methods look comparable on the 2-dimensional set, but the pivot-based method quickly establishes itself as the quicker as the data size increases. The pivot-based method becomes increasingly superior as the dimensionality continues to rise. Because the pivot-based algorithm becomes the optimal choice at the very beginning

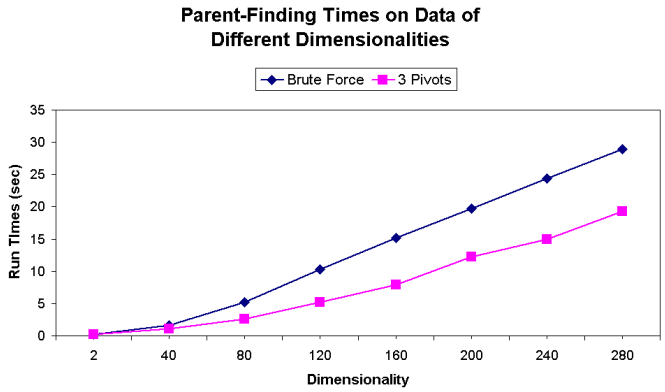


Figure 13: Run times for the brute force and pivot-based parent-finding methods on 5000 item data sets of 2-280 dimensions.

of the figure, it is also worth considering Figure 14, where we expand the first section of Figure 13 to show dimensionality ranging from 2-20.

These results may be beneficial in terms of automated algorithm selection within the HIVE framework. In [32] it was described how the system can create an appropriate algorithm, given the cardinality and dimensionality statistics of the input data. It would be simple to incorporate additional information on the optimal parent-finding routine for a particular dimensionality, and this would only serve to provide users with more efficient algorithms.

4.8 The relationship between complexity and run times

It is worth emphasising that although the parent-finding stage has been shown to be the computational bottleneck in the hybrid model, it does not necessarily follow that it is the most time-consuming stage.

Figure 15 illustrates this point. The horizontal bars represent the time spent

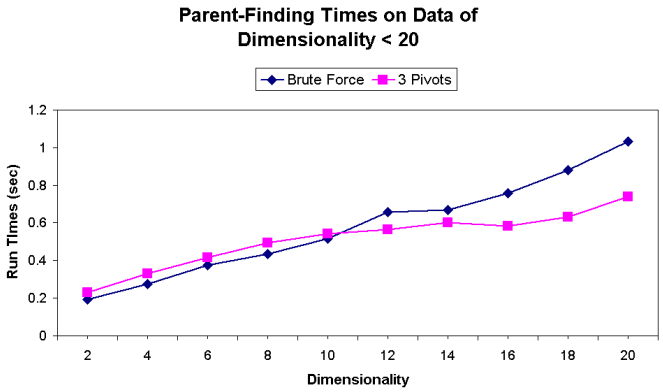


Figure 14: Run times for the brute force and pivot-based parent-finding methods on 5000 item data sets of 2-20 dimensions.

performing interpolation. Each bar is divided to show the proportions of time spent in parent-finding (left) and object placement (right). It is apparent that the object placement stage is the most time-consuming of the model. Although running in linear time, the constants selected are sufficiently large to result in longer run times than the $O(N^{\frac{5}{4}})$ parent-finding stage for data sets of this size. It may also be seen, however, that as the data size increases, the proportion of time spent on parent-finding also rises. As the size of data set continues to increase, it is likely that there will come a point where the more computationally complex stage overtakes the linear routine. We are beginning some initial test runs with larger sets of data to determine where this crossover occurs.

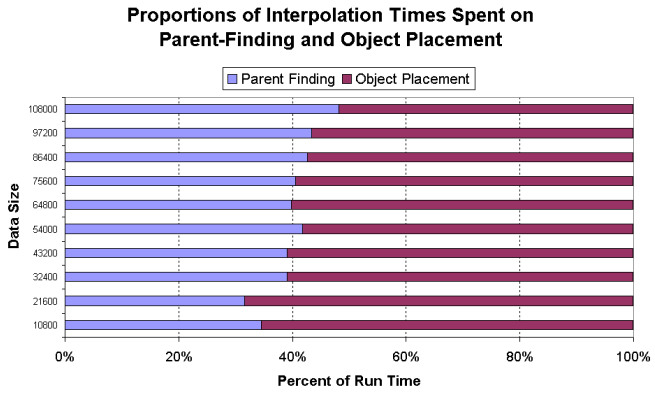


Figure 15: Horizontal bars are divided to represent the proportion of time spent on parent-finding and object placement during interpolation.

5 Conclusions and Future Work

This paper has examined the most computationally expensive phase of the hybrid MDS algorithm of [27], namely parent-finding. We proposed a novel pivot-based technique for parent-finding, and compared it against both random sampling and the original brute force method. Algorithmic analysis shows that the technique lowers the computational complexity of the 2002 algorithm: $O(N^{\frac{5}{4}})$ rather than $O(N^{\frac{3}{2}})$.

We also carried out some pilot experiments involving two data sets, which confirmed that the technique offers lower run times than its predecessor and produces good quality layouts in terms of stress. Our results also suggest that parent-finding becomes a more time-consuming part of the layout process as data sets get larger, and so the benefits of our algorithm should also increase with larger data sets.

Although our algorithm has been shown to perform well on the data described in this paper, we aim to further assess performance on data sets of varying size, dimensionality and distribution. In addition, it is our intention to further evaluate each stage of our model via a series of comparisons with alternative techniques. For example, we aim to investigate the effects on time and layout quality when different MDS models are chosen as the pre-interpolation phase of the algorithm. It would also be interesting to perform comparisons between our parent-finding routine and alternative near-neighbour algorithms.

Overall, we have tried to make the most of the hybrid approach to algorithmic design, examining and profiling not just the overall algorithm but its components. Since large data sets with complex interrelationships are of increasing concern to scientists in many domains, we suggest that this kind of algorithm and this kind of algorithmic development can make a useful contribution to large-scale information visualisation.

6 Acknowledgements

We thank Mark Baillie, Stefan Egglestone, Matt Williams and Tamara Munzner for providing data sets and we thank Greg Ross for assistance and discussion during algorithmic development.

This previous version of this paper appears in the proceedings of the IEEE Symposium on Information Visualization 2003 [25]. This extended version includes around 50% novel material.

References

[1] N. Amenta and J. Klinger. Visualizing sets of evolutionary trees. In *Proceedings of the IEEE Symposium on Information Visualization 2002*, pages 71–74. IEEE, IEEE Computer Society, 2002.

[2] K. Andrews, W. Kienreich, V. Sabol, J. Becker, G. Droschl, F. Kappe, M. Granitzer, P. Auer, and K. Tochtermann. The infosky visual explorer: Exploiting hierarchical structure and document similarities. *Information Visualization*, 1(3/4):166–181, 2002.

[3] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[4] M. Baillie and J. M. Jose. Audio-based event detection for sports video. In *Proceedings of the International Conference of Image and Video Retrieval*. Springer, 2003.

[5] I. Borg and P. J. F. Groenen. *Modern Multidimensional Scaling Theory and Applications*. Springer-Verlag, New York, 1997.

[6] J. Bourgain. On lipschitz embedding of finite metric spaces into hilbert space. *Israel Journal of Mathematics*, 52:46–52, 1985.

[7] S. Brin. Near neighbor search in large metric spaces. In *Proceedings of the 21st Conference on Very Large Databases*, pages 574–584, 1995.

[8] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, 1973.

[9] J. D. Carroll and J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of 'eckart-young' decomposition. *Psychometrika*, 35:283–319, 1970.

[10] M. Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. In *Proceedings of IEEE Visualization 1996*, pages 127–132. IEEE, IEEE Computer Society Press, 1996.

[11] Chávez E., J. L. Marroquín, and G. Navarro. Fixed queries array: A fast and economical data structure for proximity searching. *Multimedia Tools and Applications*, 14(2):113–135, 2001.

[12] Chávez E., G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[13] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd Conference on Very Large Data Bases*, pages 426–435, 1997.

[14] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[15] A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In R. Tamassia and I. G. Tollis, editors, *Proc. DIMACS Int. Work. Graph Drawing, GD*, pages 388–403, Berlin, Germany, 10–12 1994. Springer-Verlag.

[16] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[17] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of 25th International Conference on Very Large Data Bases*, pages 518–529, 1999.

[18] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM Press, 1998.

[19] Y. Koren, L. Carmel, and D. Harel. Ace: A fast multiscale eigenvectors computation for drawing huge graphs. In *Proceedings of the IEEE Symposium on Information Visualization 2002*, pages 137–144. IEEE, IEEE Computer Society, 2002.

[20] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[21] J. B. Kruskal and M. Wish. *Multidimensional Scaling*. Sage Publications, Beverly Hills, CA, 1978.

[22] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematics and Probability*, pages 281–297. Berkeley, University of California Press, 1967.

[23] S. Maneewongvatana and D. M. Mount. On the efficiency of nearest neighbor searching with data clustered in lower dimensions. In *International Conference on Computational Science*, pages 842–851, 2001.

[24] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, 1969.

[25] A. Morrison and M. Chalmers. Improving hybrid mds with pivot-based searching. In *Proceedings of the IEEE Symposium on Information Visualization 2003*, pages 85–90. IEEE Computer Society, 2003.

[26] A. Morrison, G. Ross, and M. Chalmers. Combining and comparing clustering and layout algorithms. Technical Report 148, Department of Computing Science, University of Glasgow, November 2002.

[27] A. Morrison, G. Ross, and M. Chalmers. A hybrid layout algorithm for subquadratic multidimensional scaling. In *Proceedings of the IEEE Symposium on Information Visualization 2002*, pages 152–158. IEEE, IEEE Computer Society, 2002.

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

[29] E. Oja. A simplified neuron model as a principal component analyzer. 15:267–273, 1982.

[30] L. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition*. Prentice-Hall, Inc., 1993.

[31] K. Rodden, W. Basalaj, D. Sinclair, and K. Wood. Does organisation by similarity assist image browsing? In *Proceedings of the SIGCHI conference*

on Human Factors in Computing Systems, pages 190–197. ACM Press, 2001.

[32] G. Ross and M. Chalmers. A visual workspace for constructing hybrid multidimensional scaling algorithms and coordinating multiple views. *Information Visualization*, 2(4), 2003.

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

[34] R. N. Shepard. Multidimensional scaling with an unknown distance function. i. *Psychometrika*, 27(2):125–140, 1962.

[35] W. S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17:401–419, 1952.

[36] E. Vidal. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognition Letters*, 4:145–157, 1986.

Figure Captions:

Figure 1. 2002 Algorithm. Complexities are given in square brackets.

Figure 2. The procedure to place object i onto the sample layout. The object's 'parent' x is determined, and a circle is defined, centred on x. The radius r is equal to the high-dimensional distance between x and i. Object i's position is then determined via a minimisation of Eq. (3); the summed discrepancy between high and low dimensional distances. An ideal location on the circumference of the circle, i_c, is determined via a binary search, then a series of composite force vectors are added to give the final location i_f.

Figure 3. Diagram of one pivot object (represented by the shaded point). A pivot has a certain amount of buckets, shown as numbered discs between the dotted circles. Each remaining data item is stored in one bucket, as determined by its proximity to the pivot.

full width (both columns) Figure 4. An implementation of the algorithm in the HIVE framework. The Multiple Runs module (top left) controls a batch job of executions under different conditions and with different data sets. Clock and Stress modules measure performance and dump results to files. The algorithm is running on the 17-dimensional Antarctic data.

Figure 5. Time taken to complete each of three methods of parent-finding on 13-dimensional data sets of sizes 10800-108000 objects. The graph displays mean results of six runs performed on each size using each model.

Table 1. Accuracy of parents found, as determined by rank in list of nearest neighbours.

Figure 6. 2-dimensional layout of an approximately evenly-distributed data set with imposed clustering. Point A illustrates a worst-case example of distance from the parent, point B a best case.

Figure 7. Post-interpolation stress levels across different data sizes with 3 different parent-finders.

Figure 8. Total times for layout generation with each of the parent-finding methods executing on the 13D audio data.

Figure 9. Total times for layout generation of the 17D Antarctic data set.

Table 2. Final stress values for each parent-finding method, on both data sets. Stress values, as expected, are very similar at the end of execution, as our algorithm terminates automatically when layout stability is achieved.

colour: Figure 10. Completed layout of audio data using hybrid MDS algorithm. Each point represents one second of sound. The clusters labelled A and B correspond to speech, while C represents music.

colour: Figure 11. Layouts of 17D Antarctic data using hybrid algorithm (left) and neural PCA (right).

colour: full width (both columns) Figure 12. Selecting objects in the PCA plot (right) highlights the corresponding objects in the hybrid algorithm output. The hybrid algorithm has identified two separate groups at the extremities of the layout. PCA has positioned the same objects in a dense region, not

To form a layout of N multivariate objects :

1. Select $\sqrt{N}$ subset of objects $[O(\sqrt{N})]$
2. Create 2D layout of subset using Chalmers' [10] linear per iteration spring model $[O(N)]$
3. Interpolate remaining objects onto the layout $[O(N\sqrt{N})]$
   (a) Find parent in sample for each remaining object $[O(N\sqrt{N})]$
   (b) Use high-dimensional distances to a $N^{\frac{1}{4}}$ sample (of the sample) to position remaining objects $[O(N)]$
4. Fine-tune layout with a constant number of iterations of Chalmers' spring model run on the full data set $[O(N)]$

distinguishing separate clusters.

Figure 13. Run times for the brute force and pivot-based parent-finding methods on 5000 item data sets of 3-280 dimensions.

Figure 14. Run times for the brute force and pivot-based parent-finding methods on 5000 item data sets of 2-20 dimensions.

Figure 15. Horizontal bars are divided to represent the proportion of time spent on parent-finding and object placement during interpolation.

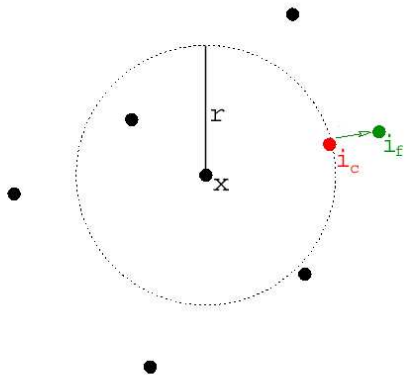
Figures:

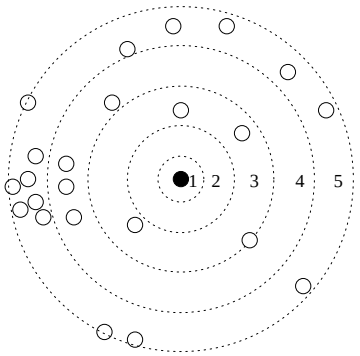
The number of figures mentioned in the text: 15 , of which 3 colour: Figures 10, 11 and 12. If possible, we would like figures 4 and 12 to be full width (straddle both columns).

The number of figures attached:15

The number of tables mentioned in the text: 2

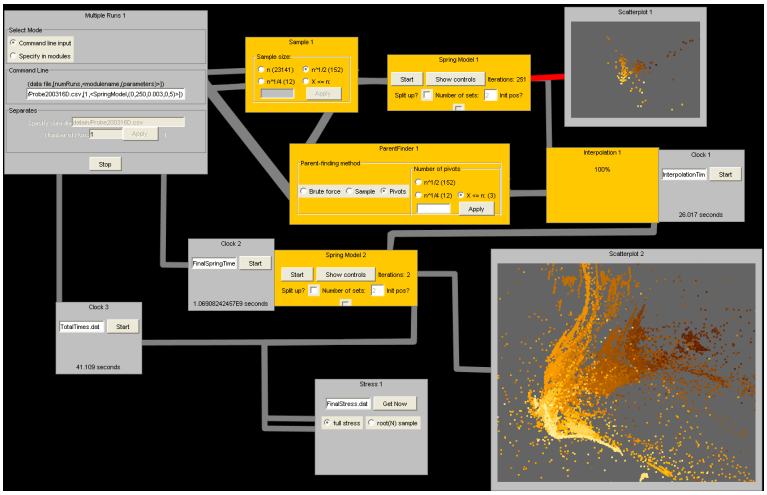
The number of figures attached:2

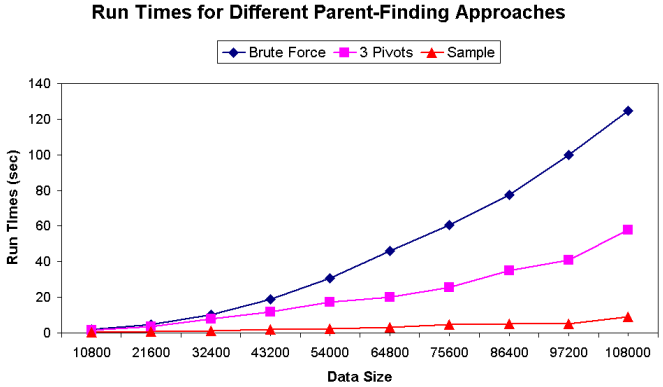


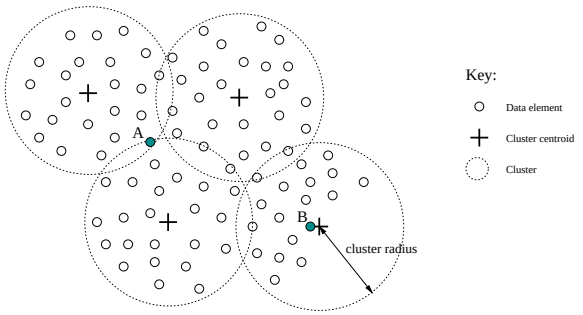


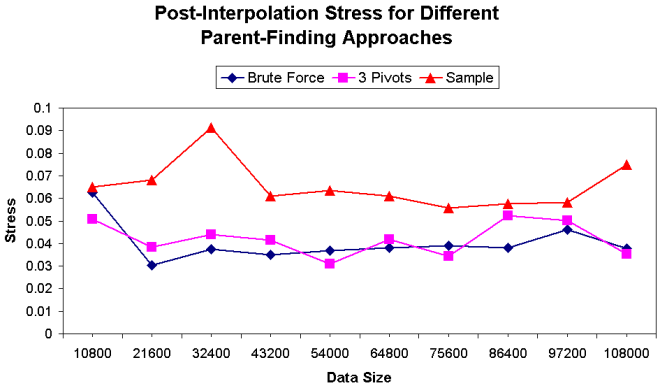
Method	Rank
Brute Force	30
Sample	110
1 pivot	59
2 pivots	37
3 pivots	35
4 pivots	34
10 pivots	32
Random	486

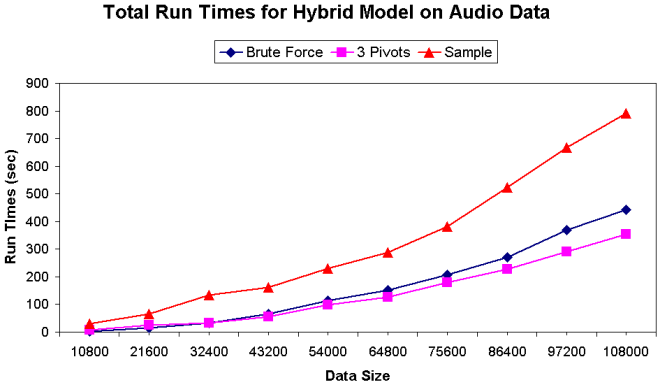
Method	Audio data	Antarctic data
Brute Force	0.02831	0.2512
Sample	0.02849	0.2481
3 pivots	0.02834	0.2562

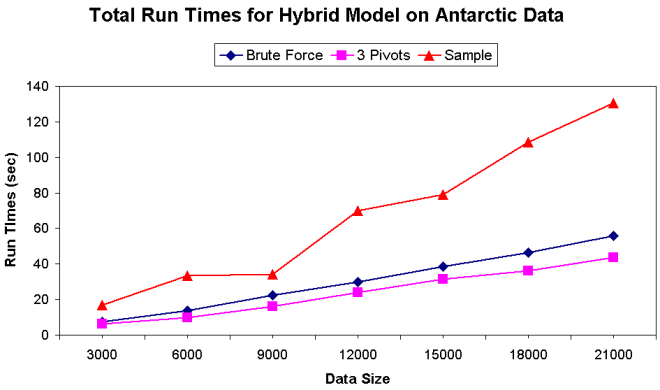

Multiple Runs 1
Select Mode
Command line input
Specify in modules
Command Line
Separates
Stop
Sample 1
Sample size:
n (23141)
n^1/2 (152)
n^1/4 (12)
X <= n
Apply
Spring Model 1
Start
Show controls
Iterations: 251
Split up?
Number of sets
Init pos?
Scatterplot 1
ParentFinder 1
Parent-finding method
Brute force
Sample
Pivots
Number of pivots
n^1/2 (152)
n^1/4 (12)
X <= n (3)
Apply
Interpolation 1
100%
Clock 1
InterpolationTim
Start
26.017 seconds
Clock 2
FinalSpringTime
Start
1.06908242457E9 seconds
Spring Model 2
Start
Show controls
Iterations: 2
Split up?
Number of sets
Init pos?
Scatterplot 2
Clock 3
TotalTimes.dat
Start
41.109 seconds
Stress 1
FinalStress.dat
Get Now
full stress
root(N) sample

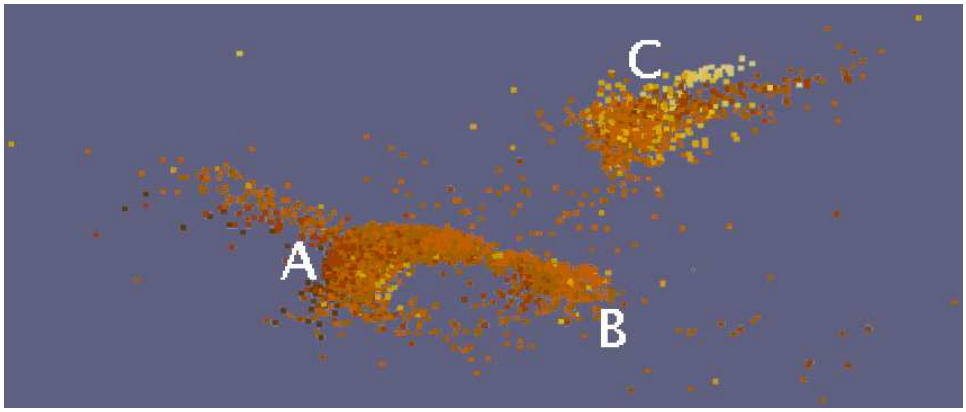

Run Times for Different Parent-Finding Approaches
Brute Force
3 Pivots
Sample
Run Times (sec)
140
120
100
80
60
40
20
0
10800
21600
32400
43200
54000
64800
75600
86400
97200
108000
Data Size

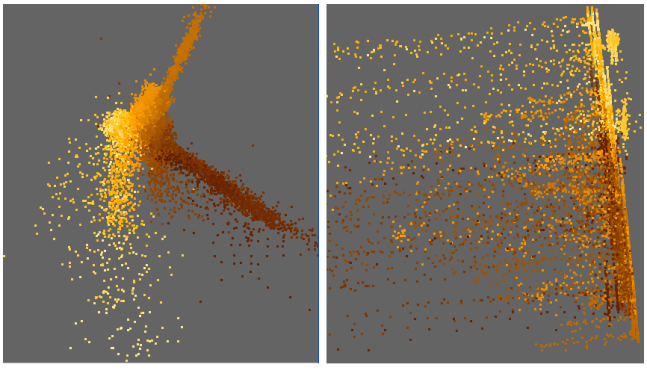

A
B
cluster radius
Key:
Data element
Cluster centroid
Cluster

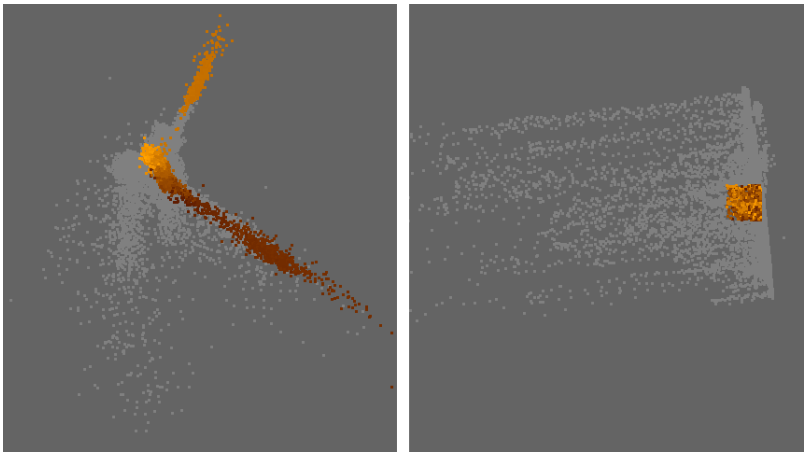

Post-Interpolation Stress for Different Parent-Finding Approaches
Brute Force
3 Pivots
Sample
Stress
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
10800
21600
32400
43200
54000
64800
75600
86400
97200
108000
Data Size

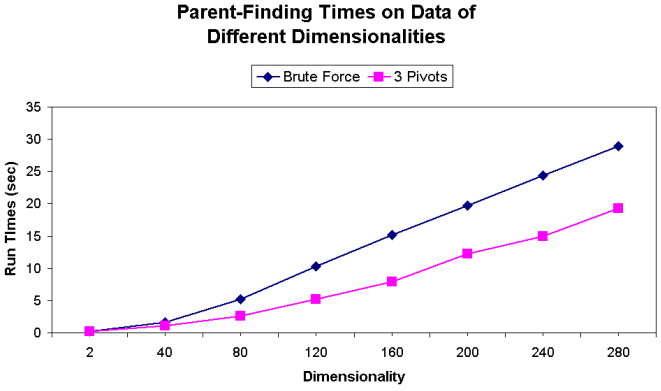

Total Run Times for Hybrid Model on Audio Data
Brute Force
3 Pivots
Sample
Run Times (sec)
900
800
700
600
500
400
300
200
100
0
10800
21600
32400
43200
54000
64800
75600
86400
97200
108000
Data Size

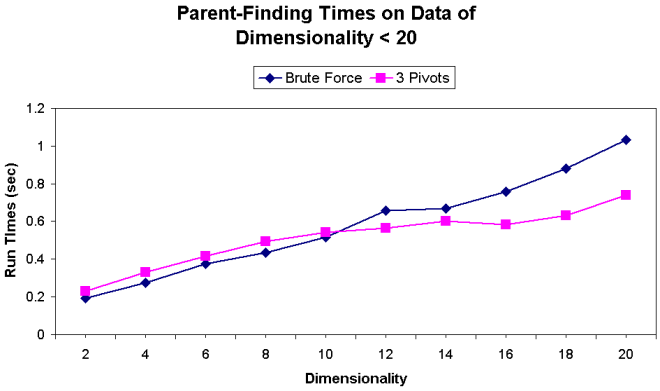

Total Run Times for Hybrid Model on Antarctic Data
Brute Force
3 Pivots
Sample
Run Times (sec)
140
120
100
80
60
40
20
0
3000
6000
9000
12000
15000
18000
21000
Data Size

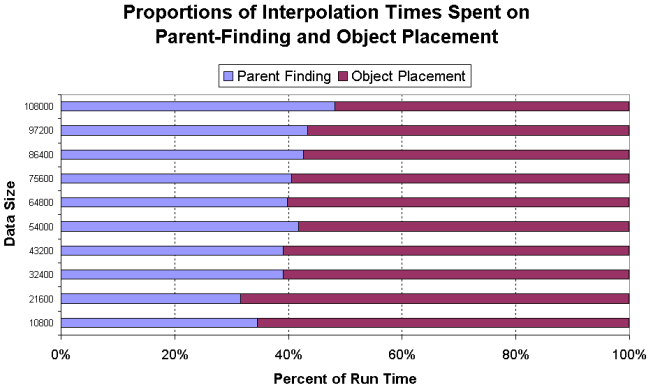

C
A
B






Parent-Finding Times on Data of Different Dimensionalities
Brute Force
3 Pivots
Run Times (sec)
35
30
25
20
15
10
5
0
2
40
80
120
160
200
240
280
Dimensionality


Parent-Finding Times on Data of Dimensionality < 20
Brute Force
3 Pivots
Run Times (sec)
1.2
1
0.8
0.6
0.4
0.2
0
2
4
6
8
10
12
14
16
18
20
Dimensionality


Proportions of Interpolation Times Spent on Parent-Finding and Object Placement
Parent Finding
Object Placement
Data Size
108000
97200
86400
75600
64800
54000
43200
32400
21600
10800
0%
20%
40%
60%
80%
100%
Percent of Run Time