FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets

*Christos Faloutsos**
Institute of Systems Research
and Dept. of Computer Science
Univ. of Maryland, College Park

King-Ip (David) Lin
Dept. of Computer Science
Univ. of Maryland, College Park

Abstract

A very promising idea for fast searching in traditional and multimedia databases is to map objects into points in k-d space, using k feature-extraction functions, provided by a domain expert [Jag91]. Thus, we can subsequently use highly fine-tuned spatial access methods (SAMs), to answer several types of queries, including the 'Query By Example' type (which translates to a range query); the 'all pairs' query (which translates to a spatial join [BKSS94]); the nearest-neighbor or best-match query, etc.

However, designing feature extraction functions can be hard. It is relatively easier for a domain expert to assess the similarity/distance of two objects. Given only the distance information though, it is not obvious how to map objects into points.

This is exactly the topic of this paper. We describe a fast algorithm to map objects into points in some k-dimensional space (k is user-defined), such that the dis-similarities are preserved. There are two benefits from this mapping: (a) efficient retrieval, in conjunction with a SAM, as discussed before and (b) visualization and data-mining: the objects can now be plotted as points in 2-d or 3-d space, revealing potential clusters, correlations among attributes and other regularities that data-mining is looking for.

We introduce an older method from pattern recognition, namely, *Multi-Dimensional Scaling* (MDS) [Tor52]; although unsuitable for indexing, we use it as yardstick for our method. Then, we propose a much faster algorithm to solve the problem in hand, while in addition it allows for indexing. Experiments on real and synthetic data indeed show that the proposed algorithm is significantly faster than MDS, (being linear, as opposed to quadratic, on the database size N), while it manages to preserve distances and the overall structure of the data-set.

1 Introduction

The objective of this work is to provide a retrieval and visualization tool for large collections of traditional as well as 'exotic' and multimedia datasets. An excellent idea, suggested by Jagadish [Jag91], was to

*This work was partially supported by the National Science Foundation (IRI-8958546 and IRI-9205273), with matching funds from Empress Software Inc. and Thinking Machines Inc.

rely on domain experts to derive k feature-extraction functions, thus mapping each object into a point in k-dimensional space. Then the problem is reduced to storing, retrieving and displaying k-dimensional points, for which there is a plethora of algorithms available.

However, it is not always easy to derive the above feature-extraction functions. Consider the case, eg., of typed English words, where the distance function is the editing distance (minimum number of insertion, deletions and substitutions to transform one string to the other). It is not clear which the features should be in this case. Similarly, in matching digitized voice excerpts, we typically have to do some time-warping [SK83], which makes it difficult to design feature-extraction functions.

Overcoming these difficulties is exactly the motivation behind this work. Generalizing the approach by Jagadish, we try to map objects into k-dimensional points, assuming that a domain expert has only provided us with a distance/dis-similarity function $\mathcal{D}(*, *)$. Notice that this setting includes the case of features, by using eg., the Euclidean distance between two feature vectors as the distance function between the corresponding objects.

Given such a set of objects and the distance function $\mathcal{D}()$, users would like (a) to find objects similar to a given query object, (b) to find the pairs of objects that are most similar to each other, as well as (c) to visualize the distribution of objects into some appropriately chosen space, in order to check for clusters and other regularities.

Next, we shall use the following terminology:

Definition 1 The k-dimensional point P_i that corresponds to the object O_i, will be called 'the *image*' of object O_i. That is, $P_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,k})$

Definition 2 The k-dimensional space containing the 'images' will be called *target space*.

Some of the applications that motivated the present work are listed next. Some distance functions are also described.

- Image and, in general, multimedia databases: In a collection of shapes [Jag91] we would like to find shapes similar to a give one; in a collection of color images, we would like to find images with similar colors, shapes or texture [NBE+93]. There we used the Euclidean distance between appropriately selected feature vectors (color attributes, moments of inertia for shape, etc.) Search-by-content is highly desirable in multimedia databases, with audio (voice, music), video etc. [NC91]. For example, users might want to retrieve, music scores, or video clips that are similar to a target music score or video clip. Once the similarity (or dis-similarity) function has been determined, our proposed method can be immediately applied.
- Medical databases, where 1-d objects (eg., ECGs), 2-d images (eg., X-rays) and 3-d images (eg., MRI brain scans) [ACF+93] are stored. Ability to retrieve quickly past cases with similar symptoms would be valuable for diagnosis, as well as for medical teaching and research purposes. Notice that the distance functions are complicated, typically requiring some warping of the two images, to make sure that the anatomical structures (eg., bones) are properly aligned, before we consider the differences [TBS90]. This warping makes it difficult to find features that would adequately describe each image (and therefore, map it into a point in feature space).

- Time series, with, eg. financial data, such as stock prices, sales numbers etc., or scientific databases, with time series of sensor data, weather [CoPES92], geological, environmental, astrophysics [Vas93] data, etc., In such databases, typical queries would be '*find companies whose stock prices move similarly*', or '*find past days in which the solar magnetic wind showed patterns similar to today's pattern*' [Vas93]. The goal is to aid forecasting, by examining similar patterns that may have appeared in the past. In [AFS93] we used the Euclidean distance (sum of squared errors) as the distance function between two time series.
- Similarity searching in string databases, as in the case of spelling, typing [Kuk92] and OCR error correction [JSB91]. There, given a wrong string, we should search a dictionary to find the closest strings to it. Conceptually identical is the case of approximate matching in DNA databases, where there is a large collection of strings from a four-letter alphabet (A,G,C,T); a new string has to be matched against the old strings, to find the best candidates [AGM$^+$90]. In all these applications, the distance is typically the *editing distance* ie., minimum number of insertions, deletions or substitutions that are needed to transform the first string to the second.
- Data mining [AS94], [AIS93] and visualization applications. For example, given records of patients (with attributes like gender, age, blood-pressure etc.), we would like to help the physician detect any clusters, or correlations among symptoms, demographic data and diseases.

From the above descriptions, two types of queries seem to be very desirable: 'query-by-example' requests and 'all pairs' queries. Specifically:

Definition 3 The term *query-by-example* (or, equivalently 'range query' or 'similarity query') will signify queries of the following form: Given a desirable object (termed *query object*), *search a collection of objects to find the ones that are within a user-defined distance ϵ from the query object.*

Definition 4 The term *all pairs* query (or, equivalently 'spatial join') will signify queries of the form: In a collection of objects, *find the pairs of objects which are within distance ϵ from each other.* Again, ϵ is user-defined.

All the above applications would benefit by a mapping of objects into points in some k-d space. Such a mapping provides two major benefits:

1. It can accelerate the search time for queries. The reason is that we can employ highly fine-tuned Spatial Access Methods (SAMs), like the R^*-trees [BKSS90] and the z-ordering [Ore86]. These methods provide fast searching for range queries as well as spatial joins [BKSS94].
2. it can help with visualization, clustering and data-mining: Plotting objects as points in k=2 or 3 dimensions can reveal much of the structure of the dataset, such as the existence of major clusters, the general shape of the distribution (linear versus curvilinear versus Gaussian) etc.. These observations can provide powerful insights in formulating hypotheses and discovering rules.

Thus, as discussed before, the general problem is defined as follows. We shall refer to it as the '*distance case*', to highlight the fact that only the distance function is known:

General Problem ('distance' case)

Given N objects and distance information about them (eg., an $N \times N$ distance matrix, or simply the distance function $\mathcal{D}(*,*)$ between two objects)
Find N points in a k-dimensional space,
such that the distances are maintained as well as possible.

We expect that the distance function $\mathcal{D}()$ is non-negative, symmetric and obeys the triangular inequality. In the 'target' (k-d) space, we typically use the Euclidean distance, because it is invariant under rotations. Alternative distance metrics could be any of the L_p metrics, like the L_1 ('city-block' or 'Manhattan' distance).

A special case is when we have already extracted features from the objects, but we still want to do a projection, usually because the features are too many ('dimensionality curse'). We shall refer to it as the *'features' case*:

Specialized Problem ('features' case)

Given N vectors with n attributes each,
Find N vectors in a k-dimensional space,
such that the distances are maintained as well as possible.

Again, the distance between two vectors in either of the two spaces could be any L_p metric. As before, we choose to use the Euclidean distance (L_2 metric).

In the above problems, the ideal mapping should fulfill the following requirements:

1. It should be fast to compute: $O(N)$ or $O(N \log N)$, but not $O(N^2)$ or higher, because the cost will be prohibitive for large databases.
2. It should preserve distances, leading to small discrepancies (low '*stress*' - see (Eq. 1)).
3. It should provide a very fast algorithm to map a new object (eg., a query object) to its image. The algorithm should be $O(1)$ or $O(\log N)$. This requirement is vital for 'queries-by-example'.

The outline of this paper is as follows. In section 2 we present a brief survey of Multi-Dimensional Scaling (MDS), related dimensionality reduction methods (K-L, SVD etc) and pointers to literature on clustering and spatial access methods. In section 3 we present our method. In section 4 we give some experimental results on real and synthetic datasets. In section 5 we list the conclusions.

2 Survey

Here we present some background information about older attempts to solve the problem. First we discuss the *Multidimensional Scaling* (MDS) method that has been used in several diverse fields (eg., social sciences, psychology, market research, physics [You87]) to solve the 'distance' case problem. Then, we present the Karhunen-Loève transform and the closely related Singular Value Decomposition that has been used for dimensionality reduction ('features' case). Finally, we provide a brief survey of spatial access methods, as well as pointers to clustering algorithms.

2.1 Multi-Dimensional Scaling (MDS)

Multidimensional scaling (MDS) is used to discover the underlying (spatial) structure of a set of data items from the (dis)similarity information among them. There are several variations, but the basic method (eg., see [KW78]) is described next. Following the 'distance' case setting, the method expects (a) a set of N items, (b) their pair-wise (dis)similarities and (c) the desirable dimensionality k.

Then, the algorithm will map each object to a point in a k dimensional space, to minimize the *stress* function:

$$stress = \sqrt{\frac{\sum_{i,j} (\hat{d}_{ij} - d_{ij})^2}{\sum_{i,j} {d_{ij}}^2}} \tag{1}$$

where d_{ij} is the dissimilarity measure between object O_i and object O_j and $\hat{d}_{ij}$ is the (Euclidean) distance between their 'images' P_i and P_j. The 'stress' function gives the relative error that the distances in k-d space suffer from, on the average.

To achieve its goal, MDS starts with a guess and iteratively improves it, until no further improvement is possible. In its simplest version, the algorithm works roughly as follows: It originally assigns each item to a k-d point (eg., using some heuristic, or even at random). Then, it examines every point, computes the distances from the other $N-1$ points and moves the point to minimize the discrepancy between the actual dissimilarities and the estimated k-d distances. Technically, MDS employs the method of 'steepest descent' to update the positions of the k-d points. Intuitively, it treats each pair-wise distance as a 'spring' between the two points; then, the algorithm tries to re-arrange the positions of the k-d points to minimize the 'stress' of the springs.

The above version of MDS is called *metric* multidimensional scaling [Tor52], because the distances are given as numbers. Several generalizations and extensions have been proposed to the above basic algorithm: Kruskal [KW78] proposed a method that automatically determines a good value for k; Shepard [She62], and Kruskal [Kru64] proposed the *non-metric* MDS where the distance between items are specified qualitatively; Young [You87] describes the *individual difference* MDS, which incorporates multiple distance measures, corresponding to different observers' perception of the data's difference.

MDS has been used in numerous, diverse applications, including the following: semantic structure analysis of words; perceived personality trait relationships [RSN72], operating on 60 different personality traits and people's perception of what goes together (like 'warm' and 'trusting'); physics (nuclear gamma-ray spectra pattern recognition, recognizing the different type of spins and their relationships); political science (determining ideological shifts) [You87]; texture analysis [RL92].

However, for our applications, MDS suffers from two drawbacks:

- It requires $O(N^2)$ time, where N is the number of items. Thus, it is impractical for large datasets. In the applications presented above, the number of items was small (typically, N=10-100).
- Its use for fast retrieval is questionable: In the 'query-by-example' setting, the query item has to be mapped to a point in k-d space. MDS is not prepared for this operation: Given that the MDS algorithm is $O(N^2)$, an incremental algorithm to search/add a new item in the database would be $O(N)$ at best. Thus, the complexity of answering a query would be as bad as sequential scanning.

The above two drawbacks are the motivation behind this present paper. Despite the above problems, we use MDS as a yardstick, against which we measure the speed and 'stress' of our method.

2.2 Dimensionality reduction techniques

In the 'features' case, the problem has been studied extensively in statistical pattern recognition and matrix algebra. The optimal way to map n-dimensional points to k-dimensional points ($k \leq n$) is the *Karhunen-Loève* ('K-L') transform (eg., see [DH73], [Fuk90]). K-L is optimal in the sense that it minimizes the mean square error, where the error is the distance between each n-d point and its k-d image.

Figure 1 shows a set of 2-d points, and the corresponding 2 directions (x' and y') that the K-L transform suggests: If we are allowed only k=1, the best direction to project on is the direction of x'; the next best is y' etc.

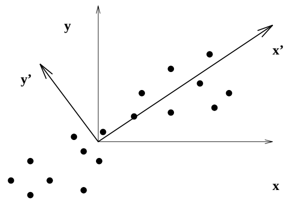


Figure 1: Illustration of the Karhunen-Loève transformation - the 'best' axis to project is x'.

'K-L' is often used in pattern matching [Fuk90] to choose the most important features (actually, linear combinations of features), for a given set of vectors. It computes the eigenvectors of the covariance matrix, sorts them in decreasing eigenvalue order, and approximates each data vector with its projections on the first k eigenvectors. The operation is closely related to the Singular Value Decomposition (SVD) [Str80, PFTV88, GV89] of the object-feature matrix. Appendix A gives our implementation of the K-L transform in *Mathematica* [Wol91].

However, the K-L transform suffers from two drawbacks:

- it can **not** be applied at all on the 'distance' case
- even in the 'features' case, it may be slow for large databases ($N \gg 1$) with many attributes ($n \gg 1$)

The latter situation appears, eg., in information retrieval and filtering [FD92], [Dum94], where documents correspond to V-dimensional vectors (V being the vocabulary size of the collection, typically in the tens of thousands). Sub-section 3.3 provides such an example.

2.3 Retrieval and Clustering

As mentioned before, the retrieval engine will be a Spatial Access Method (SAM), which, by definition, is a method that can handle k-dimensional points, rectangles, or even more complicated shapes. The most popular methods form three classes: (a) tree-based methods like the R-tree [Gut84], and its variants (R^+-tree [SRF87], hB-tree [LS90], P-tree [Jag90a], R^*-tree [BKSS90], Hilbert R-trees [KF94]

Symbols	Definitions.
N	Number of objects in database
n	dimensionality of original space ('features' case only)
k	dimensionality of 'target space'
$\mathcal{D}(*,*)$	the distance function between two objects
$\|\|\vec{x}\|\|_2$	the length ($= L_2$ norm) of vector $\vec{x}$
(AB)	the length of the line segment AB

Table 1: Summary of Symbols and Definitions

etc.) (b) methods using linear quadtrees [Gar82] or, equivalently, the z-ordering [Ore86, Ore90], or other space-filling curves [FR89, Jag90b] and finally (c) methods that use grid-files [NHS84, HN83].

There are also retrieval methods for the case where only the triangular inequality holds [BK73], [Sha77], [SW90], [BYCMW94]. All these methods try to exploit the triangular inequality in order to prune the search space on a range query. However, none of them tries to map objects into points in 'target space', nor to provide a tool for visualization.

Finally, our work could be beneficial to research on clustering algorithms, where several approaches have been proposed. See, eg., [Mur83], [Har75] for surveys, [NH94] for a recent application in GIS, [SM83] [VR79] for applications in Information Retrieval.

3 Proposed Method

In the first part, we describe the proposed algorithm, which achieves a fast mapping of objects into points, so that distances are preserved well. Then, we give an arithmetic example with a small distance matrix, and a larger example with real data. Table 1 lists the symbols and their definitions.

3.1 Algorithm

The goal is to solve the problem for the 'distance' case, that is, to find N points in k-d space, whose Euclidean distances will match the distances of a given $N \times N$ distance matrix. The key idea is to pretend that objects are indeed points in some unknown, n-dimensional space, and to try to project these points on k mutually orthogonal directions. The challenge is to compute these projections from the distance matrix only, since it is the only input we have.

For the rest of this discussion, an object will be treated as if it were a point in an n-d space, (with unknown n).

The heart of the proposed method is to project the objects on a carefully selected 'line'. To do that, we choose two objects O_a and O_b (referred to as *'pivot objects'* from now on), and consider the 'line' that passes through them in n-d space. The algorithm to choose pivot objects is discussed later (see Figure 3.1).

The projections of the objects on that line are computed using the *cosine law*. See Figure 2 for an illustration.

Theorem 1 (Cosine Law) In any triangle $O_a O_i O_b$, the cosine law gives:

$$d_{b,i}{}^2 = d_{a,i}{}^2 + d_{a,b}{}^2 - 2x_i d_{a,b} \tag{2}$$

Proof: From the Pythagorean theorem in the two rectangles $O_a E O_i$ and $O_b E O_i$.

Eq. 2 can be solved for x_i, the first coordinate of object O_i:

$$x_i = \frac{d_{a,i}{}^2 + d_{a,b}{}^2 - d_{b,i}{}^2}{2d_{a,b}} \tag{3}$$

In the above equations, d_{ij} is a shorthand for the distance $\mathcal{D}(O_i, O_j)$ (for $i, j = 1, \ldots, N$. Notice that the computation of x_i only needs the distances between objects, which are given.

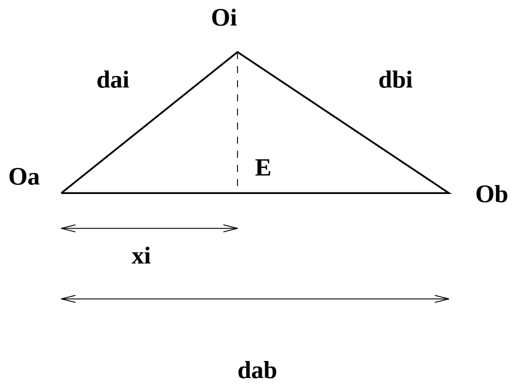


Figure 2: Illustration of the 'cosine law' - projection on the line $O_a O_b$.

Observe that, thanks to Eq. 3, we can map objects into points on a line, preserving some of the distance information: For example, if O_i is close to the pivot O_a, x_i will be small. Thus, we have *solved the problem* for k=1.

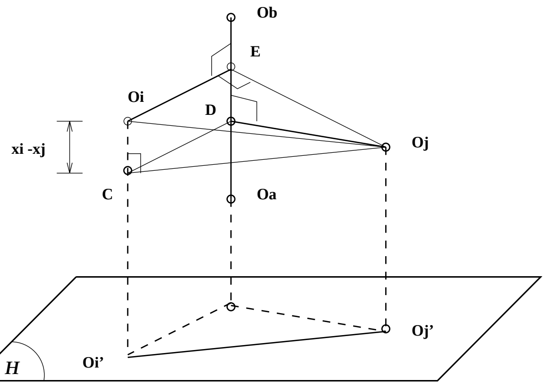


Figure 3: Projection on a hyper-plane $\mathcal{H}$, perpendicular to the line $O_a O_b$ of the previous figure.

The question is whether we can extend this method, so that we can map the objects into points in 2-d space, and eventually, k-d space. The answer is affirmative, and the idea is as follows: Pretending that the objects are indeed points in n-d space, consider a $(n-1)$-d hyper-plane $\mathcal{H}$ that is perpendicular to the line (O_a, O_b); then, project our objects on this hyper-plane. Let $O_i{}'$ stand for the projection of O_i (for $i = 1, \ldots, N$). The problem is the same as the original problem, with n and k decreased by one. This should not create problems, because n was unknown to begin with!

The only missing part is to determine the distance function $\mathcal{D}'()$ between two of the projections on the hyper-plane $\mathcal{H}$, such as, $O_i{}'$ and $O_j{}'$. Once this is done, we can recursively apply the previous steps.

Figure 3 depicts two objects O_i, O_j, and their projections $O_i{}'$, $O_j{}'$ on the $\mathcal{H}$ hyper-plane. A key observation is the next Lemma:

Lemma 1 On the hyper-plane $\mathcal{H}$, the Euclidean distance $\mathcal{D}'()$ between the projections $O_i{}'$ and $O_j{}'$ can be computed from the original distance $\mathcal{D}()$, as follows:

$$(\mathcal{D}'(O_i{}', O_j{}'))^2 = (\mathcal{D}(O_i, O_j))^2 - (x_i - x_j)^2 \quad i, j = 1, \ldots, N \tag{4}$$

Proof: From the Pythagorean theorem on the triangle $O_i C O_j$ (with the right angle at 'C') we have:

$$(O_i{}'O_j{}')^2 = (CO_j)^2 = (O_iO_j)^2 - (O_iC)^2 \tag{5}$$

where (AB) indicates the length of the line segment AB. Since $(O_iC) = (DE) = ||x_i - x_j||_2$, the proof is complete.

Ability to compute the distance $\mathcal{D}'()$ allows us to project on a second line, lying on the hyper-plane $\mathcal{H}$, and, therefore, orthogonal to the first line (O_a, O_b) by construction.

Thus, we can solve the problem for a 2-d 'target' space. More importantly, we can apply the same steps recursively, k times, thus *solving the problem for any k.*

The point that we have not discussed is how to choose the 'pivot objects' O_a and O_b. Clearly, we would like to find a line on which the projections are as far apart from each other as possible. To achieve that, we need to choose O_a and O_b such that the distance $\mathcal{D}(O_a, O_b)$ is maximized. However, this would require $O(N^2)$ distance computations. Thus, we propose the linear heuristic algorithm *choose-distant-objects()*, illustrated in Figure 3.1

Algorithm 1 *choose-distant-objects* ($\mathcal{O}$, $dist()$)
begin
1) Chose arbitrarily an object, and declare it to be the second pivot object O_b
2) set O_a = (the object that is farthest apart from O_b) (according to the distance function $dist()$)
3) set O_b = (the object that is farthest apart from O_a)
4) report the objects O_a and O_b as the desired pair of objects.
end

Figure 4: Heuristic to choose two distant objects.

All the steps in the above algorithm are linear on N. The middle two steps can be repeated a constant number of times, still maintaining the linearity of the heuristic. In all our experiments, we have 5 iterations.

Now we are ready to describe our basic algorithm. According to the problem definition ('distance' case), the algorithm accepts as input (a) a set $\mathcal{O}$ of N objects (eg., typed words, ASCII documents, color images, or n-d vectors) (b) a distance function $\mathcal{D}()$ that obeys the triangular inequality and (c) the desired number of dimensions k, and it maps the objects into points in k-d space, so that the distances are preserved as well as possible. The output vectors are written in a global variable, the $N \times k$ array **X**[]. The algorithm also records the 'pivot objects' for each recursive call, in the global $2 \times k$ array **PA**[]. Figure 3.1 gives the pseudo-code for *FastMap*.

Algorithm 2 *FastMap*
begin
  Global variables:
  $N \times k$ array **X**[ ]
  /* At the end of the algorithm, the i-th row will be the image of the i-th object. */
  $2 \times k$ pivot array **PA**[]
  /* stores the ids of the pivot objects - one pair per recursive call */
  int col# =0;
  /* points to the column of the **X**[] array currently being updated */
  Algorithm *FastMap*(k, $\mathcal{D}()$, $\mathcal{O}$)
  1) if ($k \leq 0$)
      { return; }
    else
      {col# ++;}
  2) /* choose pivot objects */
    let O_a and O_b be the result of *choose-distant-objects*($\mathcal{O}$, $\mathcal{D}()$);
  3) /* record the ids of the pivot objects */
    PA[1, col#] = a; **PA**[2, col#]= b;
  4) if ($\mathcal{D}(O_a, O_b) = 0$)
      set **X**[i, col#] =0 for every i and return
      /* because all inter-object distances are zeros */
  5) /* project the objects on the line (O_a, O_b) */
    for each object O_i,
      compute x_i using Eq. 3 and update the global array: $\mathbf{X}[i, col\#] = x_i$
  6) /* consider the projections of the objects on a hyper-plane perpendicular to the line (O_a, O_b); the distance function $\mathcal{D}'()$ between two projections is given by Eq. 4 */
    call *FastMap*($k-1$, $\mathcal{D}'()$, $\mathcal{O}$)
end

Figure 5: Algorithm '*FastMap*'

Thus, the algorithm determines the coordinates of the N objects on a new axis, after each of the k

	O1	O2	O3	O4	O5
O1	0	1	1	100	100
O2	1	0	1	100	100
O3	1	1	0	100	100
O4	100	100	100	0	1
O5	100	100	100	1	0

Table 2: Distance matrix for arithmetic example

recursive calls. Therefore, the i-th object is mapped to the point $P_i = (\mathbf{X}[i,1], \mathbf{X}[i,2], \ldots \mathbf{X}[i,k])$ where $\mathbf{X}[i,j]$ is the j-th co-ordinate P_i, the image of the i-th object.

The complexity of the '*FastMap*' algorithm is $O(Nk)$ distance calculations: At each recursive call, the longest steps are steps 2 and 5, each of which is $O(N)$.

The reason that we need to record the 'pivot objects' in each recursive call is to facilitate queries. The search algorithm is as follows: when a 'query-by-example' request arrives, the query object O_q is mapped into a k-d point in 'target space', by projecting it on the lines of appropriate 'pivot objects', with the appropriate distance function each time. That is, we repeat step 5 of the *FastMap* algorithm for the query object only.

Notice that the complexity of the mapping operation is constant ($O(1)$) with respect to the database size N. More detailed, the algorithm requires $\Theta(k)$ distance-calculation operations, because we need to compute the distance of the query object from each of the $2 * k$ pivot objects. Even if we decide to compute the distances between the pivot objects on the fly, we have to add k more distance calculations to the count, for a total of $3 * k$.

3.2 Arithmetic Example

Next we provide an arithmetic example, to illustrate how the algorithm works. Consider a set of 5 objects, with the distance matrix of Table 2. Notice that the first three objects seem to form a cluster, and so do the rest two; the two clusters are roughly 100 units apart. With the above input and k=3, the algorithm produces the output matrix $\mathbf{X}[]$, as shown in Table 3. Notice that each row corresponds to one object; columns give the coordinates of each object. The coordinates in the space created by *FastMap* will be referred to by '*FastMap*-coordinates': f_1, f_2, f_3 etc.

The j-th column of the $\mathbf{X}[]$ matrix is produced at the j-th recursive call of *FastMap*. O1 and O4 were chosen as the pivot objects in the first recursive call, because $\mathcal{D}(O1, O4)$=100 ties in the first place for length (see Table 2).

The projection of the first pivot object is always zero; the projection of the second pivot object is always its distance from the first. Thus, $\mathbf{X}$[1,1]=0, $\mathbf{X}$[4,1]=$\mathcal{D}$(O1,O4)=100. The pivot objects were (O5 , O2), and (O3, O5) in the next two recursive calls. Notice that the stress is decreasing after each recursive call, as expected. Figure 6 shows the scatter-plot, using the first two '*FastMap*-coordinates' f_1 and f_2 of the 'target' space. Notice that the two clusters already can be distinguished.

X[]	f_1	f_2	f_3
O1	0	0.707089	0.668149
O2	0.005	1.41418	0.935411
O3	0.005	1.06062	0
O4	100	0.707089	0.668149
O5	99.995	0	1

Iteration#	Pivot	Stress
1	O1, O4	0.008
2	O5, O2	0.004
3	O3, O5	0.001

Table 3: Results of FastMap: (left) 'images' of the 5 objects; (right) pivot objects and 'stress' values in each recursive call

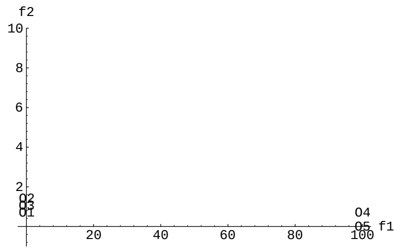


Figure 6: Scatter-plot for the 5 objects of the arithmetic example

3.3 Case Study: Document Vectors and Information Retrieval.

Here we trace the algorithm on an information retrieval application [SM83]. There, documents are represented as vectors in V-dimensional space, where V is the size of the vocabulary of the collection. For the English language, we can expect V to range from 2,000 up to and exceeding 100,000 (the vocabulary of every-day English, and the size of a very detailed dictionary, respectively [Pet80]). The coordinates of such vectors are called *term weights* and can be binary ('1' if the term appears in the document; '0' if not) or real-valued, with values increasing with the importance (eg., occurrence frequency) of the term in the document.

Consider two documents d_1 and d_2, with vectors $\vec{u_1}$, $\vec{u_2}$ respectively. The similarity between two documents is typically measured by the *cosine similarity* of their vectors [SM83]:

$$similarity(d_1, d_2) = \frac{\vec{u_1} \circ \vec{u_2}}{||\vec{u_1}||_2 . ||\vec{u_2}||_2} \tag{6}$$

where '$\circ$' stands for the inner product of two vectors and $|| * ||_2$ stands for the Euclidean norm of the vector. Clearly the cosine similarity takes values between -1 and 1. Figure 7 gives an example. There, $\cos(\theta)$ is considered as the similarity of the two vectors $\vec{u_1}$ and $\vec{u_2}$. Intuitively, the cosine similarity projects all the document vectors on the unit hyper-sphere (see vectors $\vec{u}_{1,0}$ and $\vec{u}_{2,0}$ in the figure) and measures the cosine of the angle of the projections.

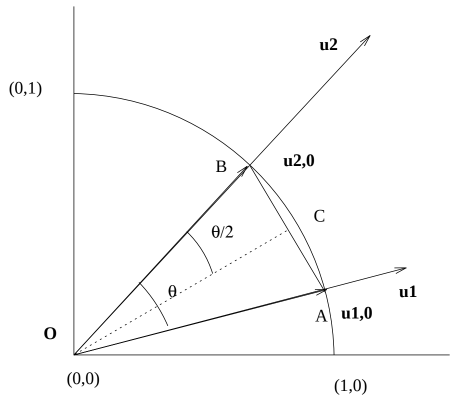


Figure 7: Two vectors $\vec{u_1}$, $\vec{u_2}$, their angle θ and the cosine similarity function $\cos(\theta)$

In order to apply our method, we first need to define a distance function that decreases with increasing similarity. From Figure 7 it would make sense to use the length of the line segment AB: $(AB) = ||\vec{u}_{1,0} - \vec{u}_{2,0}||_2$. After trigonometric manipulations, the result is

$$\begin{aligned}\mathcal{D}(d_1, d_2) &= 2 * \sin(\theta/2) \\ &= \sqrt{2 * (1 - \cos(\theta))} \\ &= \sqrt{2 * (1 - similarity(d_1, d_2))}\end{aligned} \tag{7}$$

Notice that Eq. 7 defines a distance function (non-negative, symmetric, satisfying the triangular inequality) and that it decreases with increasing similarity.

Also notice that it allows us to respond to range queries: Suppose that the user wants all the documents d that are similar to the query document q:

$$similarity(d, q) \geq \delta \tag{8}$$

Thanks to Eq. 7, the requirement becomes

$$\mathcal{D}(d, q) \leq \sqrt{2 * (1 - \delta)} \tag{9}$$

which eventually becomes a range query in our 'target' space and can be handled efficiently by any SAM.

Next we show the results of our algorithm for 10 documents (referred to as the *'DOCS-10'* dataset). The set contains 5 abstracts of computer science technical reports (labelled 'Abs1-5'), and 5 reports about basketball games (labelled 'Bbr1-5'). To create the document vector for a document, we deleted all the non-alphabetic characters, turned everything to lower-case, and deleted the most common words (using a stop-list of 150 common words). We used binary term weights, and we computed the cosine similarity among all these documents. The cosine-similarity matrix is given in Table 4. Notice that there are high similarities among the abstracts, as well as among the basketball reports, and low similarities across the two groups.

Using Eq. 7, we turned similarities into distances, and we applied '*FastMap*'. The first 5 recursive calls of the algorithm give the matrix in Table 5.

	Abs1	Abs2	Abs3	Abs4	Abs5	Bbr1	Bbr2	Bbr3	Bbr4	Bbr5
Abs1	1.000	0.230	0.117	0.162	0.174	0.000	0.013	0.000	0.000	0.000
Abs2	0.230	1.000	0.110	0.276	0.104	0.000	0.000	0.000	0.000	0.012
Abs3	0.117	0.110	1.000	0.117	0.360	0.000	0.000	0.000	0.000	0.000
Abs4	0.162	0.276	0.117	1.000	0.140	0.000	0.024	0.000	0.012	0.022
Abs5	0.174	0.104	0.360	0.140	1.000	0.000	0.000	0.000	0.000	0.000
Bbr1	0.000	0.000	0.000	0.000	0.000	1.000	0.261	0.321	0.311	0.191
Bbr2	0.013	0.000	0.000	0.024	0.000	0.261	1.000	0.307	0.326	0.354
Bbr3	0.000	0.000	0.000	0.000	0.000	0.336	0.307	1.000	0.299	0.220
Bbr4	0.000	0.000	0.000	0.012	0.000	0.311	0.326	0.299	1.000	0.237
Bbr5	0.000	0.012	0.000	0.022	0.000	0.191	0.354	0.220	0.237	1.000

Table 4: Document-to-document cosine similarity matrix for the DOCS-10 dataset

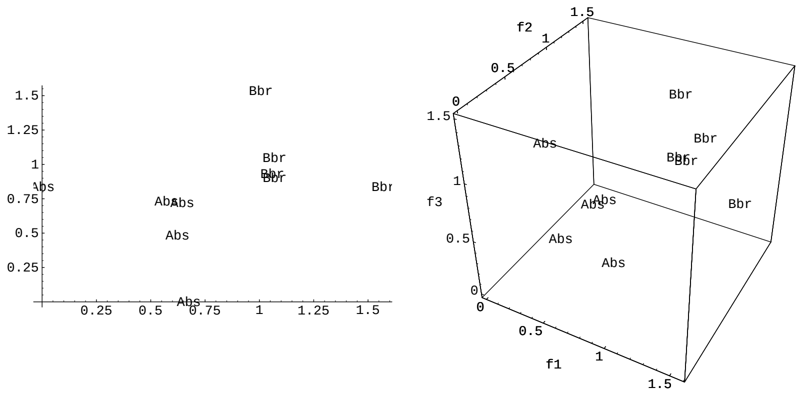
doc-Id	1st co-ord (f1)	2nd co-ord (f2)	3rd co-ord (f3)	4th co-ord (f4)	5th co-ord (f5)
Abs1	1.5708	0.730908	0.649706	0.700972	0.628742
Abs2	1.00057	0.821051	0.91341	0	0.600269
Abs3	0.898311	1.54319	0.689338	0.837133	0.543736
Abs4	0.940086	0.828169	1.48997	0.67192	0.595993
Abs5	0.950136	1.06441	0.766096	1.43761	0.600269
Bbr1	0	0.730908	0.649706	0.700972	0.628742
Bbr2	0.556887	0.482377	0.556208	0.752998	0
Bbr3	0.492452	0.611289	0	0.67192	0.595993
Bbr4	0.500826	0.594315	0.551867	0.731638	1.23218
Bbr5	0.605117	0	0.689338	0.837133	0.543736

Table 5: Output of our algorithm for the DOCS-10 dataset: 5 TR abstracts 'Abs1-5' and 5 basketball reports 'Bbr1-5'

If we keep the first 3 co-ordinates, we can plot the resulting data-set. Figure 8 gives the result (a) in 2-d space (f_1-f_2) and (b) in 3-d space. Notice that the 'images' are clustered, even in the 2-d space, agreeing with our intuition. The separation of the clusters is even better for the 3-d case.

4 Experiments

We implemented our method in 'C++' and UNIX on a DECStation 5000/25 and run several experiments, in two groups. In the first group we compared our method with the traditional MDS, with respect to speed and to quality of the output, as measured by the 'stress' function (Eq. 1). For the implementation of MDS, we use the procedure `MSIDV` from the IMSL STAT/LIBRARY FORTRAN routines.



(a) (b)

Figure 8: The DOCS-10 dataset, after *FastMap* in (a) k=2-d space (f_1 vs f_2) (b) k=3-d space. 'Abs' and 'Bbr' indicate CS TR abstracts and basketball reports, respectively.

The second group of experiments is designed to illustrate the visualization and clustering abilities of our algorithm for several applications. We used several datasets, real as well as synthetic. The real datasets are:

DOCS: It consists of 35 text documents. It includes the DOCS-10 dataset (5 CS TR abstracts (ABS), 5 basketball reports (BBR)), and in addition it has the following 5 groups (each with 5 documents):

CAL: 'Call for papers' for technical conferences.

MAT: Five portions of the Bible in King James' Version (taken from the Gospel of Matthew).

REC: Cooking recipes.

WOR: 'World News': documents about the Middle East (October 1994).

SAL: Sale advertisements for computers and software

All of the above 5 datasets are taken from various newsgroups on USENET, except for MAT, which is available electronically from `wuarchive.wustl.edu`. The distance function is based on the cosine-similarity function, as explained in subsection 3.3.

WINE N=154 records, with results of a chemical analysis of wines grown in the same region in Italy, but derived from three different cultivars. Thus, we expect to see 3 clusters. The file was obtained from the 'UC-Irvine repository of machine learning databases and domain theories' (`ics.uci.edu: //ftp/pub/machine-learning-databases/wine`). Each row has 13 attributes, indicating the amount of each of the 13 constituents found in the specific sample of wine. For the dis-similarity measure, we used the Euclidean distance, after normalizing each attribute domain to the unit interval.

The synthetic datasets are as follows:

GAUSSIAN5D We generated a dataset of N=120 points in 5-dimensional space. The points form 6 clusters, with the same number of points in each cluster. The centers of the clusters were chosen to be the points (0,0,0,0,0) (10,0,0,0,0) (0,10,0,0,0) (0,0,10,0,0) (0,0,0,10,0) (0,0,0,0,10). The data points in each cluster follow a Gaussian distribution, with standard deviation $\sigma = 1$ on each axis and covariance $\rho_{i,j} = 0$ for any $i \neq j$. Again, the distance between two such points is the Euclidean distance. This dataset is a simplified version of the one used in a Pattern Recognition textbook ([Fuk90] p. 46).

SPIRAL 30 points on a 3-d spiral, as suggested by Duda and Hart ([DH73] p.243):

$$
\begin{aligned}
x_1(i) &= \cos x_3(i) \\
x_2(i) &= \sin x_3(i) \\
x_3(i) &= i/\sqrt{2}, \quad i = 0, 1, \ldots 29
\end{aligned} \tag{10}
$$

4.1 Comparison with MDS

In the first group of experiments, we compare our method with the traditional MDS, using the 'WINE' dataset. To see the dependency on N, we run both algorithms on subsets of varying sizes, namely, N = 45, 60, 75, 90 and 105. For both methods, we experiment with k=2 and 3. Figure 9 plots the time required by each method as a function of the number of records N (a) in doubly-linear scales and (b) in doubly-logarithmic scales. We used the `time` utility of UNIX, and we report user times. In Figure 9(b) we also plotted a linear and a quadratic curve, which, in log-log scales, become straight lines with slopes 1 and 2, respectively. These lines, labelled as 'O(x)' and 'O(x^2)' respectively, are intended as visual aids, to highlight the fact that MDS requires roughly quadratic time while *FastMap* requires linear time on the database size N.

The important conclusion is that *FastMap* achieves dramatic time savings over MDS, even for small datasets.

Next, we want to study the performance of each method as the dimensionality k of the target space increases. We used the 60-point subset and we varied k from 2 to 6. Figure 10 shows the time for each method versus k, again in doubly linear and doubly logarithmic axis, ((a) and (b) respectively). Notice that the time of our method increases with k, as expected, while the time for MDS grows even faster. Again, *FastMap* provides dramatic savings in time.

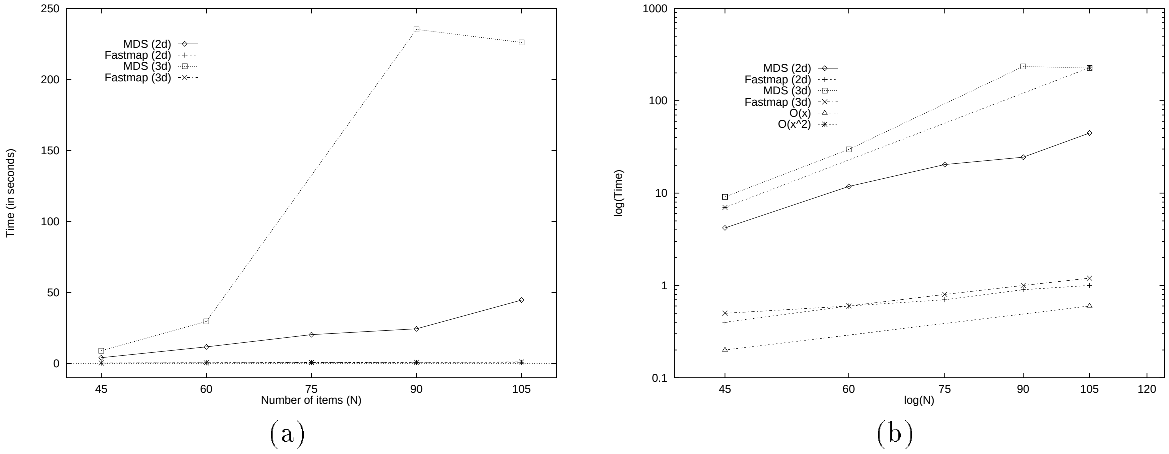
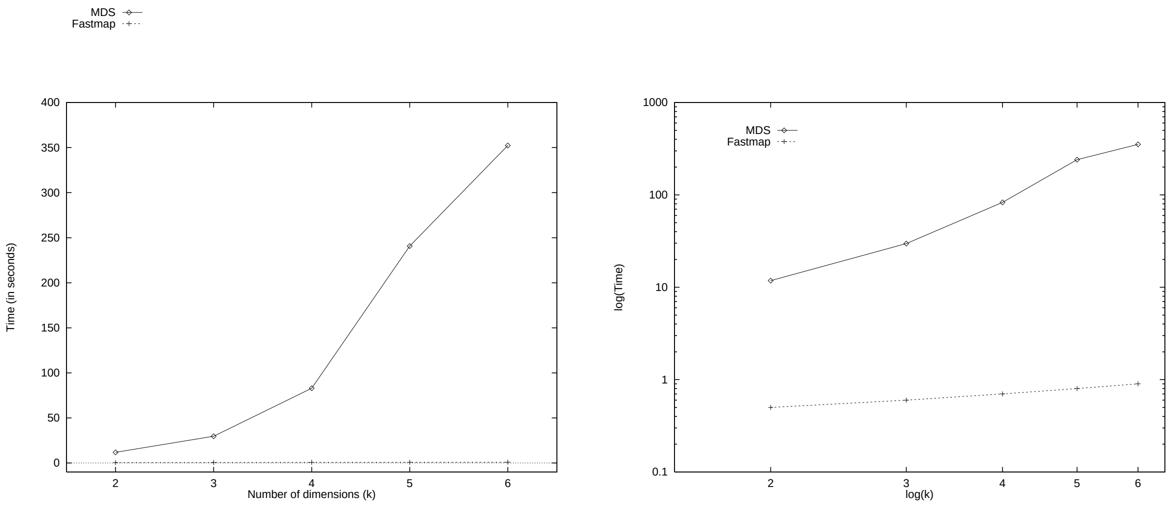
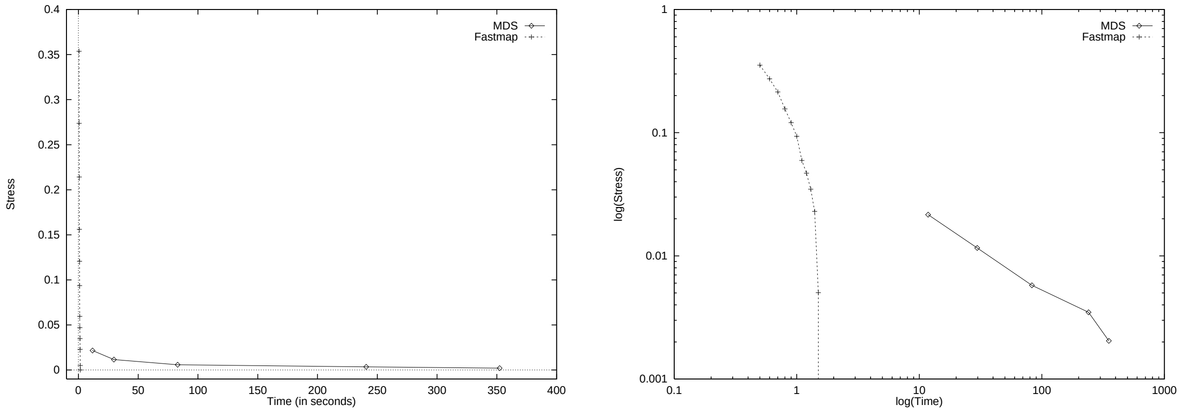


Figure 9: Response time vs. database size N for the WINE dataset; MDS and *FastMap*, with k=2,3.



(a) doubly-linear (b) doubly-logarithmic

Figure 10: Response time vs. number of dimensions k for the WINE subset (N=60) - MDS and *FastMap*



(a) doubly-linear (b) doubly-logarithmic

Figure 11: Response time vs. stress with varying k, for the WINE subset with N=60 - MDS and FastMap

The final experiment is to estimate the stress of each method. For the same dimensionality k, MDS clearly takes longer, as we saw, but it gives lower stress. The question is to find the 'price/performance' of each algorithm, that is, how much can each algorithm reduce the 'stress', in a given amount of time. Thus, Figure 11 gives the 'stress' for each method, as a function of the response time, in (a) doubly-linear and (b) doubly logarithmic axis. The independent variable was the dimensionality k. In these graphs, the 'ideal' method should give zero 'stress', in zero time. The closer a method goes to the origin (0,0), the better it is. From the doubly-logarithmic graph, we see that *FastMap* is in general closer to the 'ideal' point (0,0). Alternatively, for the same value of 'stress' (= quality), we see that *FastMap* can produce a mapping almost an order of magnitude faster.

The conclusion of this group of experiments is that, thanks to its linearity, *FastMap* achieves significant savings in time, without loss in output quality.

4.2 Clustering/visualization properties of FastMap

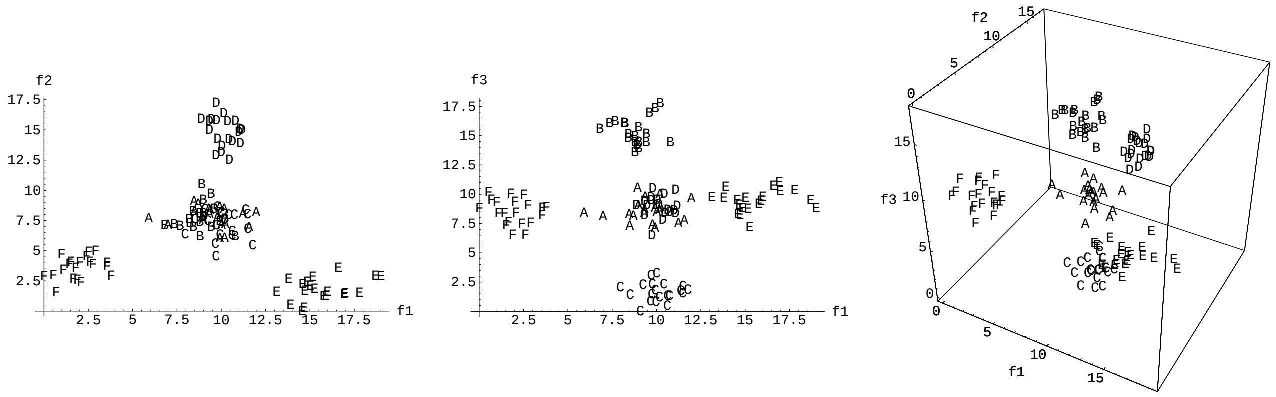
In this group of experiments our goal is to show that the proposed algorithm is useful for visualization and clustering. Here we present the results on several datasets. Unless otherwise stated, we ask for $k=3$ dimensions. Recall that f_1, f_2 and f_3 stand for the first three '*FastMap*-attributes'.

First we present the results with the synthetic datasets and then with the real ones.

4.2.1 Synthetic Data

Figure 12 gives the resulting mapping for $k=3$, for the GAUSSIAN5D dataset ($N=120$ points, forming 6 clusters, with 20 points per cluster). In the plots, the points of a given cluster are all indicated by the same letter. Figure 12(a) gives the scatter-plot of f_1 vs f_2, while (b) gives the scatter-plot of f_1 vs. f_3, and (c) gives the 3-d plot with all three '*FastMap*-attributes' Notice that, even with the first two only dimensions f_1 and f_2, we can detect roughly 4 clusters; using the next scatter-plot (b), we see that the clusters can be completely separated, because any two clusters are disjoint in at least one of the scatter-plots. Figure 12(c) confirms the previous observation, showing that all 6 clusters are disjoint in the 3-d 'target' space.

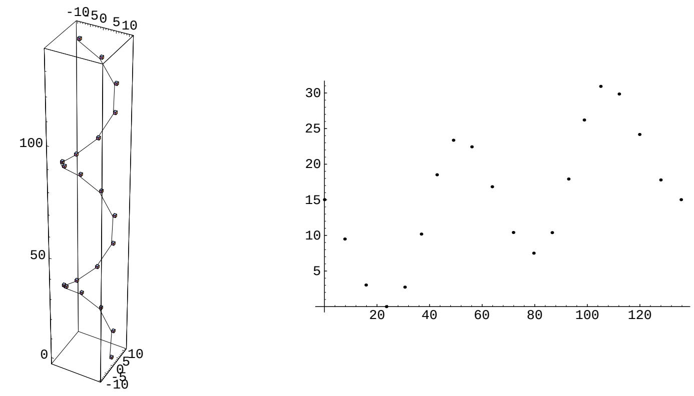
Although it uses a fictitious dataset, this example illustrates the ability of *FastMap* to help with visualization and clustering.



(a) (b) (c)

Figure 12: *FastMap* on the GAUSSIAN5D dataset (a) f_2 vs f_1 (b) f_3 vs f_1 and (c) the 3-d scatter-plot (f_1, f_2, f_3)

The next experiment involves the SPIRAL dataset. Figure 13(a) plots the original dataset in 3-d and (b) shows the result of *FastMap* for $k=2$ dimensions. Notice that the projections (Figure 13(b)) give much information about the original dataset: the points seems to form a 1-d curve, with no obvious clusters, and with some type of oscillation.



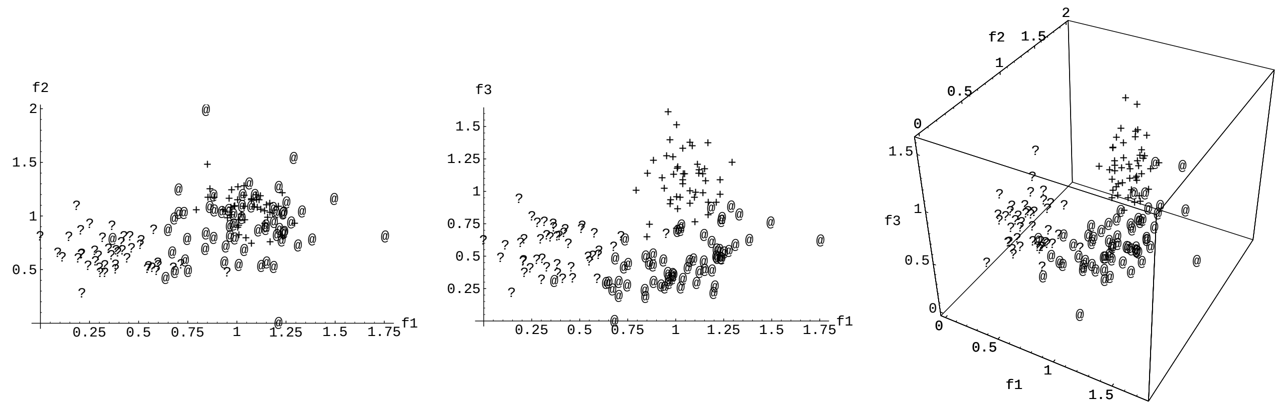
(a) (b)

Figure 13: (a) 3-d points on a spiral (SPIRAL dataset) and (b) the result of *FastMap*, for k=2

4.2.2 Real Data

Next, we present the results for the WINE dataset, in Figure 14. The layout is as in the Figure for the GAUSSIAN5D dataset: (a) gives a 2-d scatter-plot using the first two '*FastMap*-coordinates' f_1, f_2, (b) gives the scatter-plot for f_1 and f_3 and (c) combines the previous two into a 3-d scatter-plot.

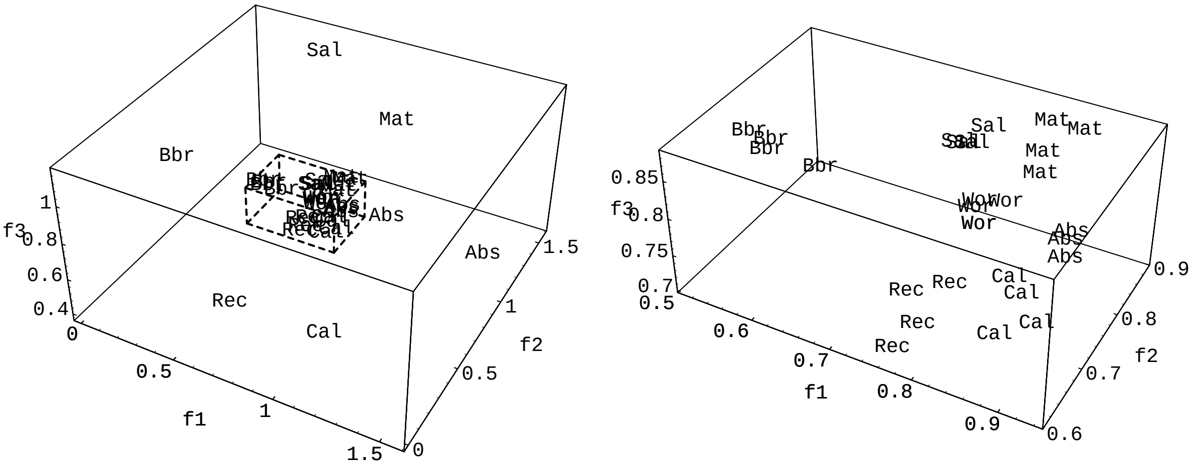
The symbols ('+', '@', '?') denote members of the first, second and third class, respectively. Notice that the f_1-f_2 scatter-plot manages to separate one of the three clusters (the one labelled with '?'). The f_1-f_3 scatter-plot provides some more information to help separate the clusters even better. The 3-d scatter-plot gives the whole picture and separates the clusters almost completely.



(a) (b) (c)

Figure 14: *FastMap* on WINE dataset (a) k=2 with f_2 vs f_1, (b) f_3 vs f_1 and (c) k=3

For our last dataset, DOCS, the results are shown in Figure 15. The figure shows the 3-d scatter-plot, (a) in its entirety and (b) after zooming into the center, to illustrate that *FastMap* manages to cluster well the documents of each class. Notice that the 7 classes are separated well, in only k=3 dimensions!



(a) (b)

Figure 15: The DOCS dataset, after *FastMap*in k=3-d space. (a) The big picture. (b) the contents of the dashed box in more detail.

5 Conclusions

We have proposed a fast algorithm to map objects into points in k-dimensional space, so that the distances between the objects are preserved as well as possible.

In an earlier approach for similarity searching in non-traditional/multimedia databases [Jag91], a domain expert was expected to provide feature extraction functions. Thanks to the proposed '*FastMap*' algorithm, the domain expert need only provide a distance function, from which our algorithm will infer the appropriate features for each object.

Mapping objects into points has the following two applications. Firstly, it can accelerate searching for several types of queries ('query-by-example' or 'range' queries, 'all pairs' queries or spatial joins [BKS93, BKSS94], nearest neighbor queries etc.), because several, highly optimized spatial access methods are readily available (R-trees [Gut84], R^*-trees [BKSS90] etc.). Secondly, such a mapping is useful for data-mining, cluster analysis and visualization of a high-dimensionality dataset.

The main contribution of this paper is the design of *FastMap*, a linear algorithm that fulfills all the design goals:

1. it solves the general problem ('distance' case) (while, eg., K-L and SVD can only solve the specialized version ('features' case))
2. it is linear on the database size, and therefore much faster than MDS and
3. at the same time, it leads to fast indexing, being able to map a new, arbitrary object into a k-d point in $O(k)$ distance calculations, regardless of the database size N.

The algorithm uses theorems from traditional geometry (such as the cosine law), and it quickly projects each object on an appropriate direction at each of the k recursive calls. With respect to quality of output (measured by the 'stress' function), we experimented with *FastMap* on real datasets: The result is that it achieves the same 'stress' levels as MDS, for a fraction of the time.

A second, smaller contribution, is Eq. 7, which turns the cosine similarity into a distance function, that can be immediately handled by *FastMap*. Our experiments on real documents showed good clustering results. Thus, coupled with Eq. 7, *FastMap* seems promising for document clustering and indexing.

The last contribution of the paper is that it introduces tools from pattern recognition, social sciences and matrix algebra, and specifically, the *Multi-Dimensional Scaling* method (MDS) and the Karhunen-Loève transform (or Singular Value Decomposition, SVD). Although not as general or as fast as the proposed algorithm, these tools could be added to the arsenal of database research, to help with indexing and visualization of non-traditional datasets. MDS has been used in diverse applications to map objects into k-d points using a quadratic, iterative algorithm. Being quadratic on N and unable to handle 'queries-by-example' easily, MDS is a good choice for visualization of small datasets. The SVD and the K-L transform provide the optimal solution for the 'features' case (although unable to handle the general problem of the 'distance' case).

Finally, we have demonstrated the speed and the output quality of our proposed algorithm on real and synthetic datasets. There, '*FastMap*' managed to separate all or most of the existing clusters, even with low values for the dimensionality k of the target space (k=2 or 3 dimensions).

Future work includes:

- Application of the algorithm to multimedia databases, where *FastMap* should *automatically* determine the features for the given dataset, from the given distance function.
- study of its benefits for interactive data mining and clustering and
- the application of the algorithm for document retrieval.

6 Acknowledgments

We would like to thank Dr. Joseph B. Kruskal from AT&T Bell Labs for providing the source code for the MDS algorithms and for answering several questions on them; Patrick M. Murphy and David W. Aha for maintaining the UC-Irvine Repository of Machine Learning Databases and Domain Theories; Prof. Howard Elman and Doug Oard for help with SVD algorithms.

A Karhunen-Loève Transform

This is the code for the K-L transform in Mathematica [Wol91]

```
(* given a matrix mat_ with $n$ vectors of $m$ attributes,
   it creates a matrix with $n$ vectors and their
   first $k$ most 'important' attributes
   (ie., the K-L expansions of these $n$ vectors) *)
KLexpansion[ mat_, k_:2] := mat . Transpose[ KL[mat, k] ];

(* given a matrix with $n$ vectors of $m$ dimensions,
   computes the first $k$ singular vectors,
```

```
   ie., the axes of the first $k$ Karhunen-Lo\'{e}ve expansion *)
KL[ mat_ , k_:2 ]:= Module[
    {n,m, avgvec, newmat,i, val, vec },

    {n,m} = Dimensions[mat];
    avgvec = Apply[ Plus, mat] / n //N;

    (* translate vectors, so the mean is zero *)
    newmat = Table[ mat[[i]] - avgvec , {i,1,n} ];

    {val, vec} = Eigensystem[ Transpose[newmat] . newmat ];
    vec[[ Range[1,k] ]]
]
```

References

[ACF+93] Manish Arya, William Cody, Christos Faloutsos, Joel Richardson, and Arthur Toga. Qbism: a prototype 3-d medical image database system. *IEEE Data Engineering Bulletin*, 16(1):38–42, March 1993.

[AFS93] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In *Foundations of Data Organization and Algorithms (FODO) Conference*, Evanston, Illinois, October 1993. also available through anonymous ftp, from olympos.cs.umd.edu: ftp/pub/TechReports/fodo.ps.

[AGM+90] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. *Proc. ACM SIGMOD*, pages 207–216, May 1993.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. *Proc. of VLDB Conf.*, pages 487–499, September 1994.

[BK73] W.A. Burkhard and R.M. Keller. Some approaches to best-match file searching. *Comm. of the ACM (CACM)*, 16(4):230–236, April 1973.

[BKS93] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient processing of spatial joins using r-trees. *Proc. of ACM SIGMOD*, pages 237–246, May 1993.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD*, pages 322–331, May 1990.

[BKSS94] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. Multi-step processing of spatial joins. *ACM SIGMOD*, pages 197–208, May 1994.

[BYCMW94] Ricardo A. Baeza-Yates, Walter Cunto, Udi Manber, and Sun Wu. Proximity matching using fixed queries trees. In M. Crochemore and D. Gusfield, editors, *5th Combinatorial Pattern Matching, LNCS 807*, pages 198–212. Springer-Verlag, Asilomar, CA, June 1994.

[CoPES92] Mathematical Committee on Physical and NSF Engineering Sciences. *Grand Challenges: High Performance Computing and Communications*. National Science Foundation, 1992. The FY 1992 U.S. Research and Development Program.

[DH73] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[Dum94] Susan T. Dumais. Latent semantic indexing (LSI) and TREC-2. In D. K. Harman, editor, *The Second Text Retrieval Conference (TREC-2)*, pages 105–115, Gaithersburg, MD, March 1994. NIST. Special Publication 500-215.

[FD92] Peter W. Foltz and Susan T. Dumais. Personalized information delivery: an analysis of information filtering methods. *Comm. of ACM (CACM)*, 35(12):51–60, December 1992.

[FR89] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. *Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 247–252, March 1989. also available as UMIACS-TR-89-47 and CS-TR-2242.

[Fuk90] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1990. 2nd Edition.

[Gar82] I. Gargantini. An effective way to represent quadtrees. *Comm. of ACM (CACM)*, 25(12):905–910, December 1982.

[Gut84] A. Guttman. R-trees: a dynamic index structure for spatial searching. *Proc. ACM SIGMOD*, pages 47–57, June 1984.

[GV89] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, second edition, 1989.

[Har75] John A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, 1975.

[HN83] K. Hinrichs and J. Nievergelt. The grid file: a data structure to support proximity queries on spatial objects. *Proc. of the WG'83 (Intern. Workshop on Graph Theoretic Concepts in Computer Science)*, pages 100–113, 1983.

[Jag90a] H. V. Jagadish. Spatial search with polyhedra. *Proc. Sixth IEEE Int'l Conf. on Data Engineering*, February 1990.

[Jag90b] H.V. Jagadish. Linear clustering of objects with multiple attributes. *ACM SIGMOD Conf.*, pages 332–342, May 1990.

[Jag91] H.V. Jagadish. A retrieval technique for similar shapes. *Proc. ACM SIGMOD Conf.*, pages 208–217, May 1991.

[JSB91] Mark A. Jones, Guy A. Story, and Bruce W. Ballard. Integrating multiple knowledge sources in a bayesian ocr post-processor. In *First International Conference on Document Analysis and Recognition*, Saint-Malo, France, September 1991. to appear.

[KF94] Ibrahim Kamel and Christos Faloutsos. Hilbert r-tree: an improved r-tree using fractals. In *Proc. of VLDB Conference,*, pages 500–509, Santiago, Chile, September 1994.

[Kru64] Joseph B. Kruskal. Nonmetric multidimensional scaling. *Psychometrika*, 29:1–27, 1964.

[Kuk92] Karen Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–440, December 1992.

[KW78] Joseph B. Kruskal and Myron Wish. *Multidimensional scaling.* SAGE publications, Beverly Hills, 1978.

[LS90] David B. Lomet and Betty Salzberg. The hb-tree: a multiattribute indexing method with good guaranteed performance. *ACM TODS*, 15(4):625–658, December 1990.

[Mur83] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983.

[NBE+93] Wayne Niblack, Ron Barber, Will Equitz, Myron Flickner, Eduardo Glasman, Dragutin Petkovic, Peter Yanker, Christos Faloutsos, and Gabriel Taubin. The qbic project: Querying images by content using color, texture and shape. *SPIE 1993 Intl. Symposium on Electronic Imaging: Science and Technology, Conf. 1908, Storage and Retrieval for Image and Video Databases*, February 1993. Also available as IBM Research Report RJ 9203 (81511), Feb. 1, 1993, Computer Science.

[NC91] A. Desai Narasimhalu and Stavros Christodoulakis. Multimedia information systems: the unfolding of a reality. *IEEE Computer*, 24(10):6–8, October 1991.

[NH94] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. *Proc. of VLDB Conf.*, pages 144–155, September 1994.

[NHS84] J. Nievergelt, H. Hinterberger, and K.C. Sevcik. The grid file: an adaptable, symmetric multikey file structure. *ACM TODS*, 9(1):38–71, March 1984.

[Ore86] J. Orenstein. Spatial query processing in an object-oriented database system. *Proc. ACM SIGMOD*, pages 326–336, May 1986.

[Ore90] J.A. Orenstein. A comparison of spatial query processing techniques for native and parameter spaces. *Proc. of ACM SIGMOD Conf.*, pages 343–352, 1990.

[Pet80] J.L. Peterson. Computer programs for detecting and correcting spelling errors. *CACM*, 23(12):676–687, December 1980.

[PFTV88] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, 1988.

[RL92] A. Ravishankar Rao and Jerry Lohse. Identifying high level features of texture perception. In *SPIE Conference*, San Jose, February 1992.

[RSN72] A. Kimball Romney, Roger N. Shepard, and Sara Beth Nerlove. *Multidimensional scaling: Theory and applications in the behavioral sciences : vol II – Applications.* Seminar Press, New York, 1972.

[Sha77] M. Shapiro. The choice of reference points in best-match file searching. *Comm. of the ACM (CACM)*, 20(5):339–343, May 1977.

[She62] R. N. Shepard. The analysis of proximities: Multidimensional scaling with an u nknown distance i and ii. *Psychometrika*, 27:125–140, 219–246, 1962.

[SK83] David Sankoff and Joseph B. Kruskal. *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparisons*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1983.

[SM83] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. The r+ tree: a dynamic index for multi-dimensional objects. In *Proc. 13th International Conference on VLDB*, pages 507–518, England,, September 1987. also available as SRC-TR-87-32, UMIACS-TR-87-3, CS-TR-1795.

[Str80] Gilbert Strang. *Linear Algebra and its Applications*. Academic Press, 1980. 2nd edition.

[SW90] Dennis Shasha and Tsong-Li Wang. New techniques for best-match retrieval. *ACM TOIS*, 8(2):140–158, April 1990.

[TBS90] A.W. Toga, P.K. Banerjee, and E.M. Santori. Warping 3d models for interbrain comparisons. *Neurosc. Abs.*, 16:247, 1990.

[Tor52] W. S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17:401–419, 1952.

[Vas93] Dimitris Vassiliadis. The input-state space approach to the prediction of auroral geomagnetic activity from solar wind variables. *Int. Workshop on Applications of Artificial Intelligence in Solar Terrestrial Physics*, September 1993.

[VR79] C.J. Van-Rijsbergen. *Information Retrieval*. Butterworths, London, England, 1979. 2nd edition.

[Wol91] Stephen Wolfram. *Mathematica*. Addison Wesley, 1991. Second Edition.

[You87] Forrest W. Young. *Multidimensional scaling : History, Theory and Applications*. Lawrence Erlbaum associates, Hillsdale, New Jersey, 1987.

Contents