

# Evaluating a Class of Dimensionality Reduction Algorithms

*Jelena Tesic*  
*jelena@iplab.ece.ucsb.edu*  
*Department of Electrical and Computer Engineering*  
*University of California at Santa Barbara*  
*Santa Barbara, CA 93106*

## Abstract

High-dimensional index structures (R\* tree, SS-tree) are used for retrieval of multimedia objects in databases. As dimensionality increases, query performance in the index structures degrades. Dimensionality reduction algorithms are the only known solution that supports scalable object retrieval and satisfies precision of query results. By combining the methods from pattern recognition, multimedia databases, and multidimensional scaling, we are trying to evaluate a set of existing dimensionality reduction algorithms that will give us satisfying performance with respect to scalability, cost, and error rate of the output. Algorithms are applied to color and texture image vectors. Generally, there are two major research directions in dimensionality reduction. The first approach is based on distance mapping algorithms while the second is based on the SVD (singular value decomposition) technique. Then, we explore more compact ways of implementing high-dimensional indexing for large datasets.

# 1 Introduction

Storage and management of multimedia objects, such as maps, video, images, text, and audio, have become an important research issue in the past couple of years. If multimedia images are similar by content to a query image, they are retrieved. This content-based retrieval can be reduced to a problem of searching in high-dimensional space if we use feature vectors to represent multimedia objects.

As a consequence, we can use different spatial access methods designed for supporting fast search in large high-dimensional databases. However, performance of similarity searching in high-dimensional index structures degrades as the dimensionality of the data increases. One solution would be to develop an indexing structure designed for high-dimensional vector space. There have been some attempts in that direction, such as X-tree, Hybrid tree, TV-tree, SS tree, and SR-tree [6], but improvement of performance in high-dimensional space is not significant. The other approach is to reduce the dimensionality so that existing indexing achieves a satisfying level of performance. Therefore, scalable multidimensional indexing can be accomplished if the dimension of the feature vector is reduced. The reduction of data dimensions introduces an error and the main problem in this approach is finding a suitable algorithm that will optimize the tradeoff between precision of a proposed algorithm and their scalability.

Significant work had been done on the eigenspace representation of objects. The underlying idea is to represent features in a transformed space where the individual features are uncorrelated. That can be achieved by performing a singular value decomposition (SVD) [5]. The SVD-based approach for dimensionality reduction condenses most of the information in a dataset to a few dimensions using eigenvectors of transformed space. This approach has two drawbacks. Since SVD is computed *a priori* on the entire set, it is not applicable on dynamic databases and it is computationally expensive. If we insert or delete a new object, SVD has to be computed for the new set of data. An incremental SVD algorithm [3] tries to lower the cost of SVD computation when the new object is inserted by introducing an approximate SVD algorithm. An SVD transform of the condensed data set (instead of the expanded one) achieves scalable performance for indexing structure update [4].

Another approach uses distance-mapping [1, 2]. A set of objects is mapped to a *k-dimensional* space in such a way that dissimilarities between objects (given distance function) are approximately preserved. Expensive distance function calculations are replaced by sum-of-square calculations. In a general case, finding a distance-preserved mapping function is a difficult and usually time-consuming process. FastMap [1] is the first algorithm proposed for fast and efficient mapping. It projects the objects to points in *k-dimensional* target space and then performs clustering on *k-dimensional* points. A mapping function uses  $2k$  pivot objects to form an axis in Euclidean *k-dimensional* space. With differently chosen pivot objects, we achieve different mappings and that could lead to a misclustering of newly inserted object. A partial upgrade of FastMap is a MetricMap algorithm [2]. Randomly picked  $2k$  objects are mapped to base vectors in  $2k-1$  dimensional space. Then, an orthogonal basis is found and, by using singular value decomposition (SVD), dimensionality of the space is reduced to *k*. This gives optimal mapping for selected  $2k$  pivot objects. If we add a new object (in the clustering algorithm), the orthogonal basis could be significantly different and that may degrade the performance (the number of misclustered objects).

Reduction of data dimensionality results in loss of information and may degrade the quality of retrieval. In this paper, we are evaluating the performance of dimensionality reduction algorithms using a texture dataset. The dimensionality of the feature vector is 48 and algorithms will be evaluated with respect to the computational complexity and quality of retrieval. The paper is organized as follows. Section 2 gives a general overview of SVD techniques and section 3 describes distance-mapping algorithms. Section 4 explains implementation issues when multidimensional dataset is used. Section 5 introduces experimental results and their summarization.

## 2 SVD-based Algorithms

Reducing data dimensionality may result in loss of information. The data should be transformed so that most of the information gets concentrated in few dimensions. The singular value decomposition (SVD) technique [5] examines the entire set of data and rotates the axis to maximize variance along the first few dimensions. In a dimensionality reduction approach, SVD achieves higher precision than other transforms.

SVD of a set A of N k-dimensional vectors is  $A = U\Sigma V^T$  where:

- A is the Nxk data matrix composed of the N k-dimensional vectors
- U is an Nxk orthonormal matrix
- $\Sigma$  is a kxk diagonal matrix of eigenvalues
- V is a kxk orthonormal basis matrix - SVD transform matrix

Computational complexity of SVD transform is  $O(Nk^2)$ . Also, in dynamic databases, the axes of target space have to be reoriented to cope with frequent insertions and deletions, or query precision will drop considerably. The query accuracy may be monitored and an SVD-recomputation triggered when the accuracy drops below an acceptable level.

## 2.1 Adaptive Eigenspace Computation

Orthogonal base of k-dimensional space changes dynamically and follows the changes in dynamic database. Principal value analysis, used for data dimensionality reduction, is expensive to compute because we end up doing SVD every time an object is inserted or deleted. Use of adaptive eigenspace computation when a new object is added to the set, whose SVD we already know, reduces the computational complexity to  $O(Nk)$ [3]. Some round-off error is introduced in this process, but the algorithm is still accurate, given the quality criteria.

Let  $U\Sigma V^T$  denote an approximate low-rank SVD of existing dataset A. Let  $A' = [A;x]$  and assume  $A = U\Sigma V^T + E$ . Our goal is to compute  $[U\Sigma V^T;x] \approx U' \Sigma' V'^T$  efficiently, where  $A' = [U\Sigma V^T;x] + [E;0]$ . It can be shown that an approximate SVD can be computed as follows:

$$\bullet \quad a = \frac{x^T - U(U^T x)}{\|x^T - U(U^T x)\|}, \quad \text{SVD transform} \begin{pmatrix} \Sigma & U^T x \\ 0 & a^T x \end{pmatrix} = \Omega \Lambda \Phi^T$$

$$\bullet \quad U' = (Ua)\Omega \quad V' = \begin{pmatrix} V & 0 \\ 0 & 1 \end{pmatrix} \Phi \quad \Sigma' = \Lambda$$

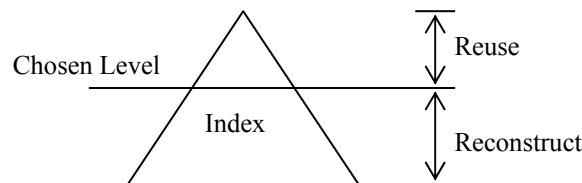
Computing SVD of broken arrowhead matrix  $\begin{pmatrix} \Sigma & U^T x \\ 0 & a^T x \end{pmatrix} = \Omega \Lambda \Phi^T$  is somewhat quicker if

we use the Gu and Eisenstat technique [5]. The proposed SVD algorithm is efficient and numerically stable and uses error-measure as the parameter for update.

## 2.2 Aggregated Eigenspace Computation

In this section, we present a different approach to the SVD transformation technique for dimensionality reduction purposes. It focuses on dynamic databases where the SVD needs to be recomputed because of changes in underlying data. Computational time for incremental SVD is still linear in the number of vectors (objects in the database). Linear dependence can be reduced if SVD computation is performed on aggregated data set [4]. The retrieval quality will depend on how well aggregated set represents the real one. This is ensured by obtaining aggregated dataset from the higher layers of an index constructed for that dataset [4]. All data items are stored at the leaf level of an index structure. We compute the centroid of each subtree and store the value at the corresponding node. This captures a good approximated distribution of data. We are choosing an aggregated dataset from the level just above the leaf level (if capacity of the node is large). Considerable reduction in SVD computation is achieved.

Incorporation of the recomputed SVD in the index entries is more complex. Our proposed approach [4] selectively reconstructs subtrees at a chosen level and reuses the existing index at higher levels (Figure 1). This *reuse-reconstruct* approach achieves optimal trade-off between high incorporation cost (if we *reconstruct* the whole index tree) and the number of index entries which can grow really high if we *reuse* nodes in the existing tree.



**Figure1.** Reuse-reconstruct strategy for incorporating SVD-transform in an existing index

### 3 Distance-Mapping Algorithms

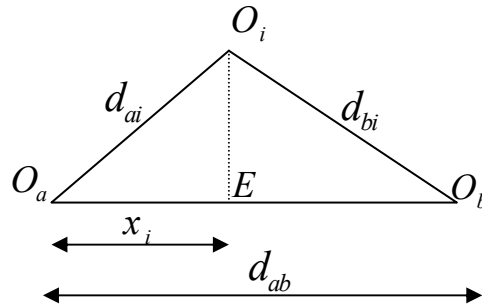
Consider a set of objects and a distance function  $d_{ij} = D(O_i, O_j)$ . Distance mapping algorithms take the set of objects and inter-object distances and map the objects to a  $k$ -d space  $R^k$  such that distances among the objects are approximately preserved. The  $k$ -d point  $P_i = (x_{i1}, x_{i2}, x_{i3}, \dots, x_{ik})$  is the image of the object  $O_i$ . Different algorithms use different methods of mapping a point into a  $R^k$  space. FastMap and MetricMap algorithms are explained in the following paragraphs.

#### 3.1 The FastMap Algorithm

This algorithm project objects on a line  $(Oa, Ob)$  in an  $k$ -dimensional space  $R^k$  [1]. The axes are formed by two pivot objects  $Oa, Ob$  for each axis, chosen as follows. First arbitrarily choose one object (let it be  $Ob$ ) and let  $Oa$  be the object that is farthest apart from  $Ob$ . Then update  $Ob$  to be the object that is farthest from  $Oa$ . The two resulting objects,  $Oa$  and  $Ob$ , are pivots.

Now consider an object  $O_i$  and the triangle formed by  $O_i, Oa$ , and  $Ob$  (Figure 2). From the cosine law we get  $d_{bi}^2 = d_{ai}^2 + d_{ab}^2 - 2x_i d_{ab}$ . Thus, the first coordinate  $x_i$  of the object  $O_i$ , where  $d_{ij} = D(O_i, O_j)$ ,

with respect to the line  $(Oa, Ob)$  is  $x_i = \frac{d_{ai}^2 + d_{ab}^2 - d_{bi}^2}{2d_{ab}}$ .



**Figure 2.** Illustration of the 'cosine law' - projection on the line  $OaOb$

We can extend the above projection method to embed the objects in target space  $R^k$  as follows.

Pretending that the given objects are indeed points in  $R^k$ , we consider an  $(k-1)$  dimensional hyperplane H that is perpendicular to the line  $(Oa, Ob)$  where  $Oa$  and  $Ob$  are pivot objects. Then, we project all of the objects onto hyperplane H. Let  $O_i$  and  $O_j$  be two objects and let,  $O_i'$  and  $O_j'$  be their projections on the hyperplane H (Figure 3). It can be shown that the dissimilarity between  $O_i'$  and  $O_j'$  is

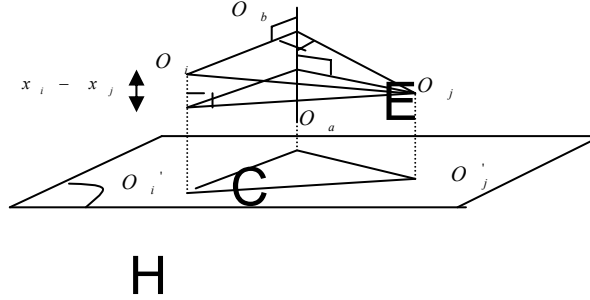
$$(D'(O_i - O_j))^2 = (D(O_i - O_j))^2 - (x_i - x_j)^2.$$

Since one can compute  $d'_{ij} = D'(O_i, O_j)$ , that allows projection on a second line, lying on the hyperplane H, and therefore orthogonal to the first line  $(Oa, Ob)$ . We repeat steps recursively, k times, thus mapping all objects to points in  $R^k$ .

The underlying assumption in this mapping assumes that the dissimilarity function is Euclidean. If that is not the case,  $(D(O_i - O_j))^2 - (x_i - x_j)^2$  can be negative. For this case we modify  $d'_{ij} = D'(O_i, O_j)$  as

$$d'_{ij} = -\sqrt{(x_i - x_j)^2 - (D(O_i - O_j))^2} \quad [2].$$

Total number of distance calculations required by FastMap is  $3Nk$  where  $N$  is the number of objects in the dataset and  $k$  is the dimension of a target space.



**Figure 3.** Projection on a hyper-plane H, perpendicular to the line  $OaOb$  of the previous figure

## 3.2 The MetricMap Algorithm

The algorithm works by choosing a small objects sample by randomly picking  $2k$  objects from the dataset[2]. SVD is applied to the sampling set in order to form an orthogonal base for the space  $R^{2k}$ .

Eigenvectors that correspond to k largest eigenvalues form the target space  $R^k$ . The algorithm then maps all objects in dataset points in  $R^k$  (Figure 4).

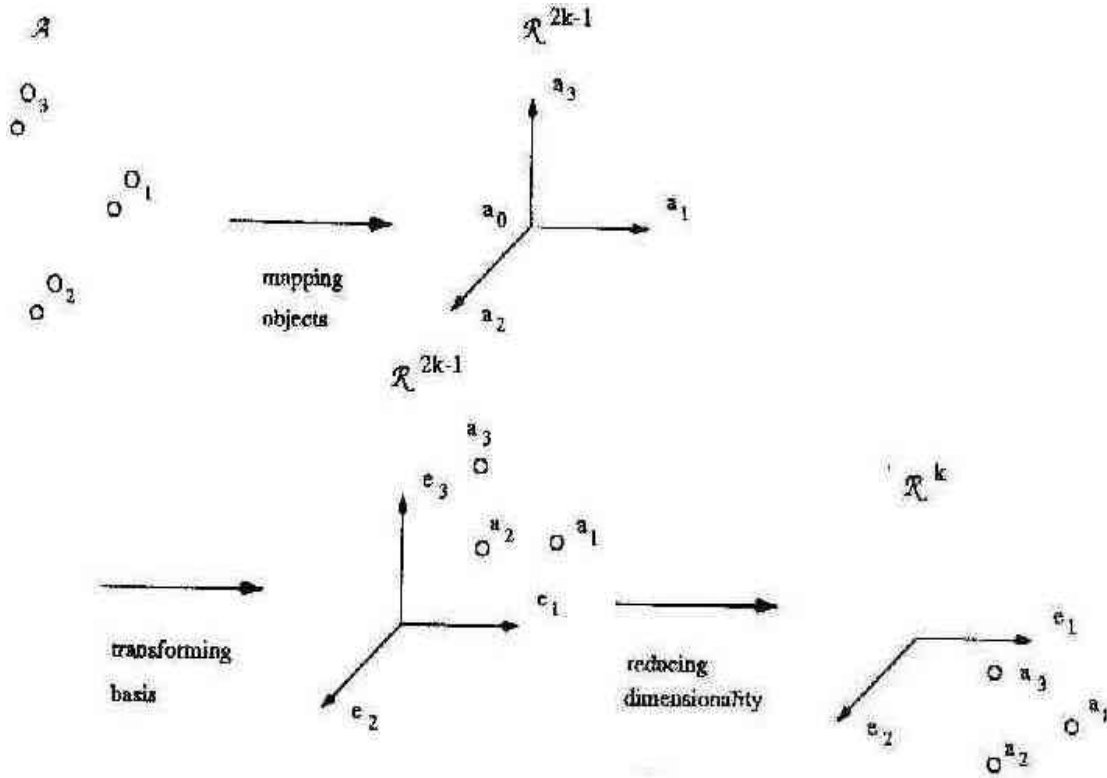


Figure 4. Illustration of MetricMap algorithm when  $k=2$

## 4 Performance Evaluation

We conducted a series of experiments to evaluate the performance of different dimensionality reduction methods. Several data structures, such as R\*-tree, SR-tree, SS-tree, TV-tree, and hybrid-tree[6], were designed to support fast searching in large multidimensional databases. So, if dimensionality reduction algorithms have a good retrieval performance, we can always incorporate an indexing structure to achieve fast query response. Our main concern is the retrieval quality.

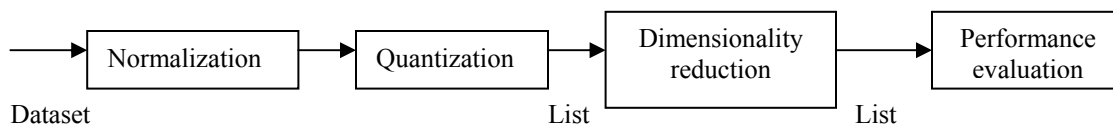
### 4.1 Dataset

We evaluate the dimensionality reduction algorithms on two datasets. The first one is a texture feature set for Brodatz album. The set consists of 1856, 50-dimensional feature vectors. They belong to 116 different classes. Each class consists of 16 vectors.

The second dataset is a feature collection of aerial photographs. These feature vectors are extracted from 40 large aerial photos. Before the feature extraction, each aerial photo is first partitioned into 64x64 non-overlapping tiles, from which the 50-dimension feature vectors are computed. That gives us 160K 50-dimensional vectors.

## 4.2 Results

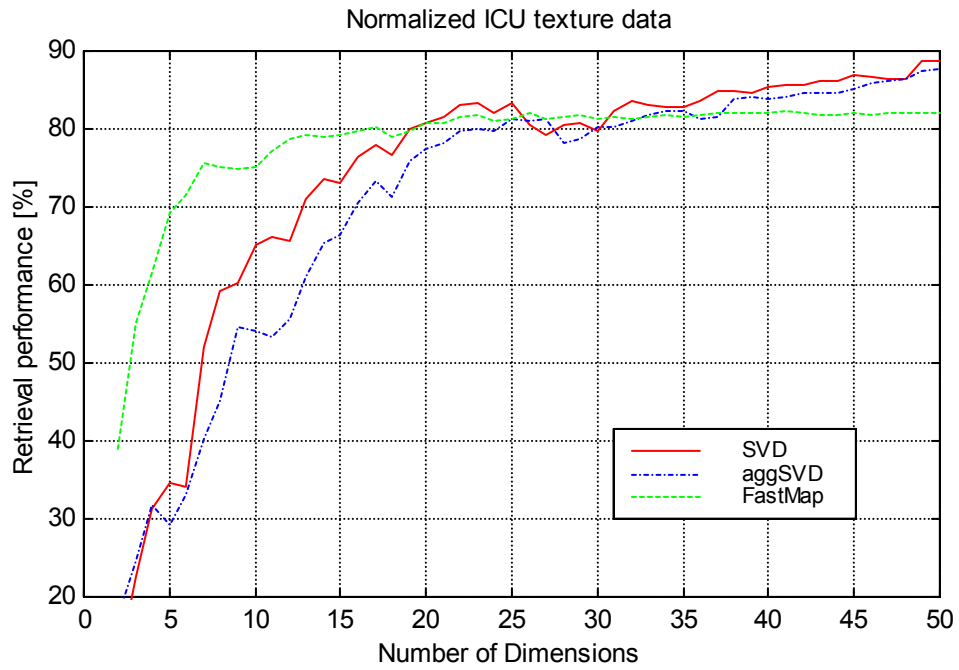
We measure the average query precision achieved when dimensionality reduction algorithms are applied to the original 50-dimensional data. Every algorithm is applied to the same dataset in order to reduce dimensionality. Then, we evaluate the retrieval precision in the object space of reduced dimensionality, as explained in Figure 5.



**Figure 5.** *Performance evaluation flow chart for dimensionality reduction algorithms*

For the Brodatz set, we have 116 classes that consist of 16 images. Retrieval precision corresponds to the number of query results that belong to the same class of images. Results are given in Table1.

In the case of aerial dataset, retrieval performance is evaluated in a different way. Images are not classified and top query matches are subjectively evaluated. Basically, we look into a number of top query results and estimate how many of them really correspond to the query image. Results are given in the Table2.



**Table 1.** Retrieval precision of dimensionality reduction algorithms, evaluated on 4K 50-dimensional texture vectors.

dimensions	Incremental SVD	Aggregated SVD	FastMap	MetricMap
4				
8				
12				
16				
20				

**Table 2.** Retrieval precision of dimensionality reduction algorithms, evaluated on 160K 50-dimensional texture vectors

dimensions	Incremental SVD	Aggregated SVD	FastMap	MetricMap
4				
8				
12				
16				
20				

## 5 Conclusion

Final results are not available and no conclusion can be drawn from implementation at this point. We hope that dimensionality reduction algorithms will work effectively for images as high-dimensional objects. However, issues like content-based representation of images and feature vector construction have a great influence on quality performance of applied dimensionality reduction techniques.

## 6 Acknowledgments

The authors would like to thank Prof. Manjunath and Prof. Singh at UCSB, for the aerial dataset and valuable comments, and to Prof. Faloutsos and Prof. Lin for the FastMap and MetricMap software.

## 7 References

- [1] C.Faloutsos and K.-I. Lin. Fast Map: A fast algorithm for indexing, data mining and visualization of traditional and multimedia datasets. *SIGMOD*, 1995.
- [2] J.Wang, X.Wang, K.-I. Lin, D. Shasha, B.Shapiro and K.Zhang. Evaluating a class of distance mapping algorithms for data mining and clustering. *ACM KDD*, 1999.
- [3] S. Chandrasekaran, B.S. Manjunath, Y.F. Wang, J. Winkeler and H.Zhang. An eigenspace update algorithm for image analysis. *CVGIP*, 1997.
- [4] K.V. Ravi Kanth, D. Agrawal and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. *ACM SIGMOD*, 1998.
- [5] G.H. Golub and C.F. van Loan. *Matrix Computations*. The John Hopkins Press, 1989.
- [6] K.-I. Lin, H.V. Jagadish and C. Faloutsos. The TV-tree: An index structure for high-dimensional data. *VLDB Journal*, 3:517-542, 1994.