RC 21755 (98001) 30 May 2000
Computer Science/Mathematics

IBM Research Report

CSVD: Clustering and Singular Value Decomposition for Approximate Similarity Searches in High Dimensional Spaces

Vittorio Castelli
IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York
e-mail: vittorio@us.ibm.com

Alexander Thomasian
CS&E Dept. at Univ. of Connecticut,
One University Place
Stamford CT 06901
e-mail: athomas@uconnvm.uconn.edu.

Chung-Sheng Li
IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York
e-mail: csli@watson.ibm.com

Research Division
Almaden – Austin – Beijing – Haifa – T.J. Watson – Tokyo – Zurich

CSVD: Clustering and Singular Value Decomposition for Approximate Similarity Searches in High Dimensional Spaces

Vittorio Castelli, Alexander Thomasian and Chung-Sheng Li

June 12, 2000

Abstract

High-dimensionality indexing of feature spaces is critical for many data-intensive applications such as content-based retrieval of images or video from multimedia databases and similarity retrieval of patterns in data mining. Unfortunately, even with the aid of the commonly-used indexing schemes, the performance of nearest neighbor (NN) queries (required for similarity search) deteriorates rapidly with the number of dimensions. We propose a method called Clustering with Singular Value Decomposition (CSVD) to reduce the number of index dimensions, while maintaining a reasonably high precision for a given value of recall. In CSVD, homogeneous points are grouped into clusters such that the points in each cluster are more amenable to dimensionality reduction than the original dataset. Within-cluster searches can then be performed with traditional indexing methods, and we describe a method for selecting the clusters to search in response to a query. Experiments with texture vectors extracted from satellite images show that CSVD achieves significantly higher dimensionality reduction than plain SVD for the same Normalized Mean Squared Error (NMSE). Conversely, for the same compression ratio, CSVD results in a decrease in NMSE with respect to simple SVD (a 6-fold decrease a 10:1 compression ratio). This translates to a higher efficiency in processing approximate NN queries, as quantified through experimental results.

1 Introduction

Similarity-based retrieval has become an important tool for searching image and video databases, especially when the search is based on content of images and videos, and is performed using low-level features, such as texture, color histogram and shape. The application areas of this technology are quite diverse, as there has been a proliferation of databases containing photographic images [34, 24, 30], medical images, [16], video clips [32], geographically referenced data [31], and satellite images [29, 22].

Efficient similarity search invariably requires precomputed features. The length of the feature vectors can be potentially large. For example, the local color histogram of an RGB image (obtained by quantizing the colors from a sliding window) has usually at least 64 dimensions, and often the dimensionality of texture vectors is 50 or more [21]. Similarity search based on features is equivalent to a *nearest neighbor* (NN) query in the high-dimensionality space of the feature vectors. NN search based on sequential scan of large files, or tables, of feature vectors is computationally too expensive and has a long response time.

Multidimensional indexing structures, which have been investigated intensively in recent years [14], seem to be the obvious solution to this problem. Unfortunately, as a result of a combination of phenomena, customarily known as the "curse of dimensionality" [9, Chapter 1], the efficiency of indexing structures deteriorates rapidly as the number of dimensions increases [5, 4, 14]. A potential solution is therefore to reduce the number of dimensions of the search space before indexing the data. The challenge is to achieve index space compression (which in general results in loss of information) without affecting information retrieval performance. In this paper, we use the terms "index space compression" and "data compression" to denote the reduction of the average number of dimensions used to describe each entry in the database.

Dimensionality reduction methods are usually based on a linear transformations followed by the selection of a subset of features, although nonlinear transformations are also possible. Techniques based on linear transformations, such as the Karhunen-Loeve (KL) transform, the Singular Value Decomposition (SVD) method, and Principal Component Analysis (PCA) [17, 12], have been widely used for dimensionality reduction and data compression [19] and form the basis of our study. Nonlinear multidimensional scaling for dimensionality reduction [2], and other fast methods, such as FastMap [13, 12], have also been investigated.

SVD is shown to be very effective in compressing large tables with numeric values in

[19], where a 40:1 compression ratio is achieved, with an average error of less than 5%. This method outperforms a clustering-based method, and a spectral method [12]. SVD

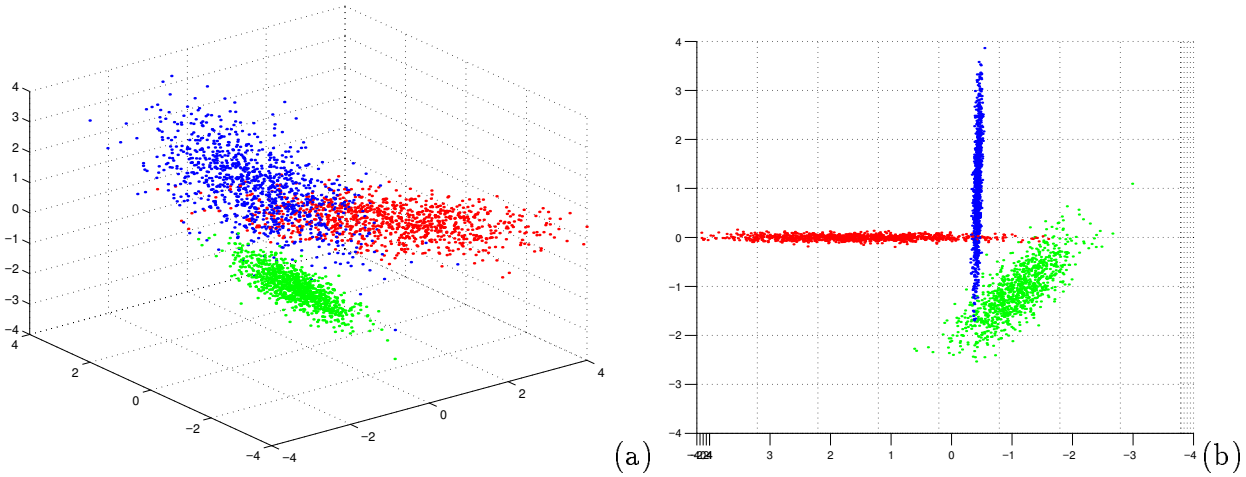

Figure 1: Intuition about using clustering to assist dimensionality reduction. The synthetic data was obtained by independent sampling from a mixture of three 3-dimensional Gaussian distributions.

relies on "global" information derived from all the vectors in the dataset. Its application is therefore more effective when the datasets consist of "homogeneously distributed vectors", or, better, of datasets the distribution of which is well captured by the centroid and the covariance matrix. This is the case discussed in Korn [19], where the authors deal with outliers by storing them separately and searching them exhaustively.

For databases with "heterogeneously" distributed vectors, more efficient representation can be generated by subdividing the vectors into groups, each of which is then characterized by a different set of statistical parameters. In particular, this is the case if the data comes from a set of populations each of which is homogeneous in the sense defined above. As a more tangible example, consider Fig. 1: here, a set of points in 3-dimensional space is clustered into three relatively flat and elongated ellipsoids. The "ellipsoids" are obtained by independent sampling from synthetically generated 3-dimensional Gaussian distributions. Figure 1(b) shows clearly that for two of the ellipsoids, most of the variance is captured by two of the three dimensions. Then, the points in each ellipsoid can be represented rather accurately in two dimensions. Note, though, that reducing the dimensionality of the entire dataset, without partitioning the data first, would result in large approximation errors. Real data can display the same behavior: consider for example the scatterplot, displayed in Figure 2(a), of three texture features extracted from a satellite

image of the Earth, or the scatterplot of Figure 2(b) showing measures acquired from a computer for capacity management purposes.

In statistics, often data is described by decomposing it into a mixture of Gaussian distributions, and methods exist to solve the estimation problem [10]. Young and Liu [35] relied on this insight to propose a method for lossless data compression, called multilinear compression, consisting of clustering followed by singular value decomposition. The assumption underlying multilinear compression is that some of the cluster actually lie on lower-dimensional subspaces. More recently, Aggarwal and Yu [1] proposed a fast method for combining clustering and dimensionality reduction.

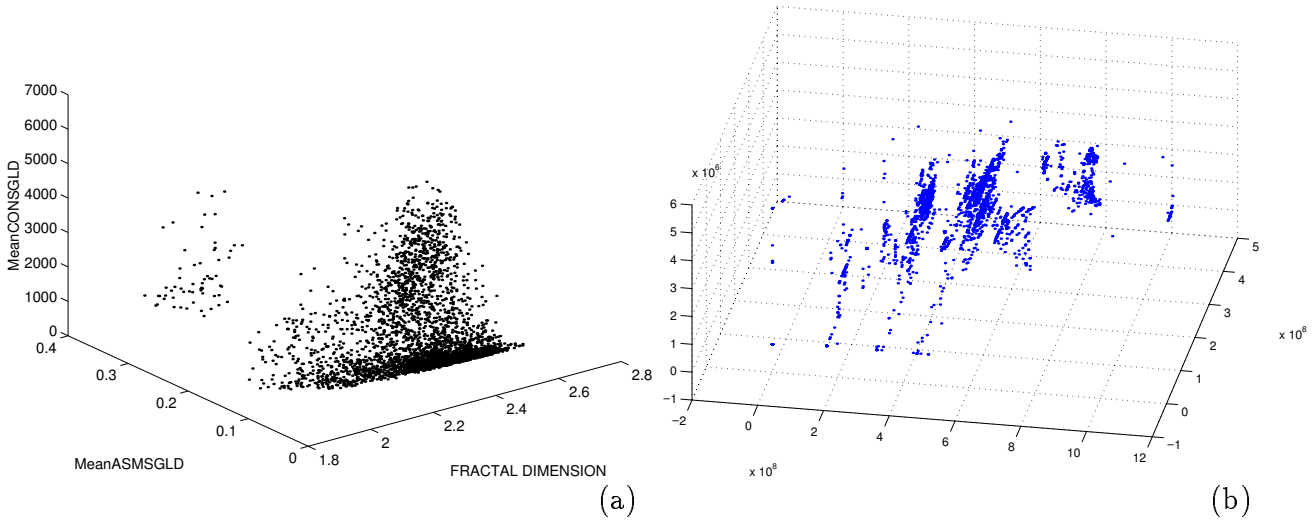

Figure 2: Scatterplot of a three-dimensional feature vector set computed from a satellite image, showing clustering behavior (a). Projections on 3-dimensional space of 4025 100-dimensional vectors of measurements acquired from a computer, for capacity management purposes. The figure shows that the distribution of the measured quantities is really a mixture of different distributions, each of which is characteristic of a particular load or class of loads (b).

This paper introduces a new method, *Clustering Singular Value Decomposition (CSVD)*, for indexing data by reducing the dimensionality of the space. This method consists of three steps: partitioning the data set using a clustering technique [11] (see Section 4), computing independently the SVD of vectors in each cluster to produce a vector space of transformed features with reduced dimensionality, and constructing an index for the transformed spaced. An iterative procedure then discovers the "optimal" degree of clustering

based on either the average tolerable error or the data compression ratio by searching.

Experiments with texture vectors from satellite images show that CSVD results in significantly better Normalized Mean Squared Error (NMSE, defined in Equations (6) and (7)) to the original data than simple SVD for the same dimensionality reduction. From the information retrieval viewpoint, experiments demonstrate that CSVD achieves better *recall* and *precision* (defined in Section 2.1) than simple SVD for the same number of retained dimensions. When used as a main-memory index, CSVD at 95% recall yields significant speedups over linear scan, and empirical data shows scalability of performance with the database size. After the reorganization performed by CSVD, the data stored in individual clusters is well suited for indexing with certain multidimensional indexing schemes. In particular we see a 3:1 to 28:1 speedup due to the within-cluster index when 20 to 30 out of 60 dimensions are retained, which is a significantly larger speedup than we would expect given the dimensionality of the search space.

The following notation is used in the paper. Lowercase bold letters denote vectors, e.g., $\mathbf{q}$, which are, unless stated, column vectors. Uppercase bold letters denote matrices. Individual elements of matrices or vectors are identified by subscripts, and are in roman faces; Thus $\mathbf{x} = [x_1, \ldots, x_n]$. $\mathbf{X}^T$ and $\mathbf{x}^T$ are the transpose of the matrix $\mathbf{X}$ and of the vector $\mathbf{x}$ respectively. To identify quantities associated to a specific cluster, say cluster h, we use a superscript between parenthesis: thus, the number of elements in cluster h will be $M^{(h)}$.

The rest of the paper is organized as follows. Section 2 contains the preliminary material on multidimensional indexing methods, and the mathematics behind SVD and PCA. The CSVD algorithm and the processing of NN queries in the new environment is described in Section 3. Section 4 describes the experimental results. Conclusions are given in Section 5.

2 Preliminaries

2.1 Indexing Methods for Nearest Neighbor Search

Typically, in similarity search an object is mapped into a point (equivalently, a vector) in a high-dimensional feature space. The similarity between two objects U and V is then measured by a function of the "distance" between the corresponding points $\mathbf{u} = [u_1, ..., u_N]^T$ and $\mathbf{v} = [v_1, ..., v_N]^T$. The $k-$NN query retrieves from the database the k

most similar entries to a specified query object, that is, the ones that are closer in the selected metric. When all the features (i.e., the entries of the vectors) are numerical, the most commonly used similarity measure is the Euclidean distance

$$D(\mathbf{u},\mathbf{v}) = \Big[(\mathbf{u}-\mathbf{v})^T(\mathbf{u}-\mathbf{v})\Big]^{1/2} = \sqrt{\sum_{i=1}^{n}(u_i - v_i)^2}. \qquad (1)$$

An NN query based on a sequential scan of the table of feature vectors is unacceptable from the viewpoint of both computational cost and response time, unless the table is representable in a very compact form via quantization [8].

Indexing of feature vectors seems to be the obvious solution, and there are many multidimensional indexing structures such as grid files, k-d-b trees [27], R-trees [15], and R*-trees [3] to choose from [12]. However, even X-trees [6], which in general provide significant performance gains over R*-trees, do not improve the execution time of NN queries [5]. Inefficiency associated with high-dimensionality indices has been recently discussed in the context of R-tree-like structures, because of their popularity [8, 4]. Similar arguments appear in [9] where classification and regression problems in high dimensions are analyzed, and in [7], which presents an interesting perspective on NN searches in high dimensions. Parallel processing solutions have been proposed to mitigate the problem [4].

Experimental results in [8] show that: "As dimensionality increases, all points begin to appear almost equidistant from one-another. They are effectively arranged in a d-dimensional sphere around the query, no matter where the query is located. The radius of the sphere increases, while the width of the sphere remains unchanged, and the space close to the query point is empty." In effect, as the dimensionality grows, almost all of the pages in the index need to be touched.

The method proposed in this paper can be used in conjunction with various indexing structures, especially R-trees for which efficient NN search methods exist [28]. The indexing structure described in [18] for the efficient processing of NN-queries is however utilized in our studies (see Section 3.4).

The efficiency of approximate indexing is quantified by the *recall* and *precision* metrics in the information retrieval literature. For each query object $\mathbf{q}$, let $A(\mathbf{q})$ denote the subset of the database containing the k most similar objects to $\mathbf{q}$. Since the search algorithm is approximate, we request more than k nearest neighbors, and let $B(\mathbf{q})$ denote the result set. Finally, let $C(\mathbf{q}) = A(\mathbf{q}) \bigcap B(\mathbf{q})$, and let $|\cdot|$ denote the number of elements in a set. Then,

- *Recall* (R) is the fraction of retrieved objects that are among the k closest objects to $\mathbf{q}$, namely

$$R = \frac{|C(\mathbf{q})|}{|A(\mathbf{q})|}. \tag{2}$$

- *Precision* (P) is the fraction of the k closest points to $\mathbf{q}$ that are retrieved by the algorithm, i.e.,

$$P = \frac{|C(\mathbf{q})|}{|B(\mathbf{q})|}. \tag{3}$$

When the retrieval algorithm is approximate, one can increase the recall by increasing the size of $B(\mathbf{q})$, but this in general reduces the precision, and vice versa.

2.2 Singular Value Decomposition

In this subsection, the well-known singular value decomposition (SVD) method is reviewed in order to establish the necessary notations for the rest of the paper. Consider an $M \times N$ matrix $\mathbf{X}$ the columns of which are correlated. Let $\boldsymbol{\mu}$ be the column mean of $\mathbf{X} = [x_{i,j}]$, $(\mu_j = (1/M)\sum_{i=1}^{M} x_{i,j}$, for $1 \leq j \leq N,)$ and let $\mathbf{1}_M$ be a column vector of length M with all elements equal to 1. SVD expresses $\mathbf{X} - \mathbf{1}_M\boldsymbol{\mu}^T$ as a product of an $M \times N$ column orthonormal matrix $\mathbf{U}$ (i.e., $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix), an $N \times N$ diagonal matrix $\mathbf{S}$ containing the singular values, and an $N \times N$ real unitary matrix $\mathbf{V}$ [26]:

$$\mathbf{X} - \mathbf{1}_M\boldsymbol{\mu}^T = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{4}$$

The columns of the matrix $\mathbf{V}$ are the eigenvalues of the covariance matrix $\mathbf{C}$ of $\mathbf{X}$, defined as

$$\mathbf{C} = \frac{1}{M}\mathbf{X}^T\mathbf{X} - \boldsymbol{\mu}\boldsymbol{\mu}^T = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T. \tag{5}$$

The matrix $\mathbf{C}$ is positive-semidefinite, hence it has N nonnegative eigenvalues and N orthonormal eigenvectors. Without loss of generality, let the eigenvalues of $\mathbf{C}$ be ordered in decreasing order, i.e., $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_N$. The trace (Σ) of $\mathbf{C}$ remains invariant under rotation, i.e., $\Sigma \stackrel{\triangle}{=} \sum_{j=1}^{N} \sigma_j^2 = \sum_{j=1}^{N} \lambda_j$. The fraction of variance associated with the j^{th} eigenvalue is λ_j/Σ. The singular values are related to the eigenvalues by: $\lambda_j = s_j^2/M$ or $s_j = \sqrt{M\lambda_j}$. The eigenvectors constitute the *principal components* of $\mathbf{X}$, hence the transformation $\mathbf{Y} = \mathbf{V}(\mathbf{X} - \mathbf{1}_M\boldsymbol{\mu}^T)$ yields zero-mean, uncorrelated features. PCA retains the features corresponding to the p highest eigenvalues. Consequently for a given p it

minimizes the *Normalized Mean Squared Error* which is denoted by NMSE, and is defined as

$$NMSE = \frac{\sum_{i=1}^{M}\sum_{j=p+1}^{N} y_{i,j}^2}{\sum_{i=1}^{M}\sum_{j=1}^{N} y_{i,j}^2} = \frac{\sum_{i=1}^{M}\sum_{j=p+1}^{N} y_{i,j}^2}{\sum_{i=1}^{M}\sum_{j=1}^{N} (x_{i,j} - \mu_j)^2} \tag{6}$$

where the column mean $\boldsymbol{\mu}$ is computed in the original reference frame.

This property makes SVD the optimum linear transformation for dimensionality reduction, when the p dimensions corresponding to the highest eigenvalues are retained. Several heuristics to select p are given in [17], e.g., retain transformed features whose eigenvalues are larger than the average Σ/M. This may be unsatisfactory, for instance when an eigenvalue slightly less than Σ/M is omitted. In this paper, p is selected so that NMSE does not exceed a threshold that yields acceptable efficiency in approximate searching.

The p columns of the $\mathbf{Y} = \mathbf{XV}$ matrix corresponding to the highest eigenvalues are retained, the remaining discarded. This is tantamount to first projecting the vectors in $\mathbf{Y}$ onto the k-dimensional hyperplane passing through $\boldsymbol{\mu}$ and spanned by the eigenvectors of $\mathbf{C}$ corresponding to the largest eigenvalues, and then representing the projected vector in the reference system having as origin $\boldsymbol{\mu}$ and as coordinate axes the eigenvectors of $\mathbf{C}$.

In the Appendix we compare the approach in this paper with that of a recent paper [19], showing that, for the same storage space and approximation error, ours is computationally more efficient.

3 The CSVD Algorithm

3.1 Preprocessing of Dataset

Before constructing the index, care should be taken to appropriately scale the numerical values of the different features used in the index. In the case of texture features (used as the test dataset for this paper) some quantities have a small range (the texture feature known as fractal dimension varies between 2 and 3), while others can vary significantly more (the dynamic range of the variance of the gray-level histogram can easily exceed 10^5).

In this paper, we assume that all the features contribute equally to determine the similarity between different vectors. Consequently, the N columns of the table $\mathbf{X}$ are

separately studentized (by subtracting from each column j its empirical mean μ_j and dividing the result by an estimator of its standard deviation $\hat{\sigma}_j$) to obtain columns with zero mean and unit variance: $x'_{i,j} = (x_{i,j} - \mu_j)/\hat{\sigma}_j, \quad 1 \leq j \leq N$ and $1 \leq i \leq M$, where the number of rows M is the number of records in the database table. Note that the studentization is done only once, on the original table $\mathbf{X}$, and it is not repeated after the clustering and dimensionality reduction steps described in the following section.

After studentization, the sum of the squared entries of the table is computed as $\mathcal{E} = \sum_{i=1}^{M} \sum_{j=1}^{N} \left(x'_{i,j}\right)^2$, which is used as denominator of equation (6) to compute the NMSE. Thus, without loss of generality, we will assume that the columns of the table $\mathbf{X}$ have zero mean and unit variance.

3.2 Index Construction

Before describing the index construction algorithm, we provide the definition of the NMSE resulting from substituting the table $\mathbf{X} = [x_{i,j}]_{M \times N}$, whose columns have zero mean, with a different table $\mathbf{X}' = [x'_{i,j}]_{M \times N}$, without changing the reference frame:

$$NMSE = \frac{\sum_{i=1}^{M} \sum_{j=1}^{N} \left(x_{i,j} - x'_{i,j}\right)^2}{\sum_{i=1}^{M} \sum_{j=1}^{N} x_{i,j}^2}. \tag{7}$$

This definition is valid for any mapping from $\mathbf{X}$ to $\mathbf{X}'$, including the one used by CSVD. The basic steps of the CSVD index construction are partitioning of the database, rotation of the individual clusters onto an uncorrelated orthogonal coordinate system, dimensionality reduction of the rotated clusters, and within-cluster index construction.

1. *The partitioning step* takes as its input the table $\mathbf{X}$ and divides the rows into H homogeneous groups, or clusters, $\mathbf{X}^{(h)}$, $1, \leq h \leq H$, having sizes $M^{(1)}, \ldots, M^{(H)}$. The number of clusters can be a parameter or can be determined dynamically. In the current implementation, classical clustering methods such as LBG [23] or K-means [11] are used. These are vector-quantization-style algorithms [23], that generate clusters containing vectors which are similar to each other in the Euclidean metric sense. It is known that these methods, albeit optimal, do not scale well with the size of the database. If the speed of the index construction is an important factor, one can use suboptimal schemes, such as tree-structured vector quantizers (TSVQ) [25], or sub-sampling schemes to limit the number of data point used to train the quantizer, i.e., to construct a partition of the database.

2. *The rotation step* is carried out independently for each of the clusters $\mathbf{X}^{(h)}$, $1 \leq h \leq H$. The corresponding centroid $\boldsymbol{\mu}^{(h)}$ is subtracted from each group of vectors $\mathbf{X}^{(h)}$, and the eigenvector matrix $\mathbf{V}^{(h)}$ and the eigenvalues $\lambda_1^{(h)}, \ldots, \geq \lambda_N^{(h)}$, (ordered by decreasing magnitude), are computed as described in Section 2.2. The vectors of $\mathbf{X}^{(h)}$ are then rotated in the uncorrelated reference frame having the eigenvectors as coordinate axes, to produce $\tilde{\mathbf{X}}^{(h)}$.

3. *The dimensionality reduction step* is performed using a procedure that considers all the clusters. To understand the effects of discarding a set of dimensions within a specific cluster (say the hth, containing $M^{(h)}$ vectors), recall that this is equivalent to replacing the vectors with their projections onto the subspace passing through the centroid of the cluster and defined by the retained eigenvectors. This approximation affects the numerator of Equation (7), which is increased by the sum of the square distances between the original points and their projections. Since the dimensions are uncorrelated, calling $\lambda_{(1)}^{(h)}, \ldots, \lambda_{(p)}^{(h)}$ the retained eigenvalues, one can easily see that the ith dimension adds $\lambda_{(i)}^{(h)} M^{(h)}$ to the numerator. We use the notation $\mathbf{Y}^{(h)}$ to denote the representation of $\mathbf{X}^{(h)}$ in the translated, rotated and projected reference frame.

   We have implemented four flavors of dimensionality reduction.

   (a) *Dimensionality reduction to a fixed number of dimensions:* the number p of dimensions to retain is provided as input; the p dimensions of each cluster corresponding to the largest eigenvalues are retained.

   (b) *Independent reduction with target* NMSE *(*TNMSE*):* clusters are analyzed independently. From each cluster h the minimum number $p^{(h)}$ of dimensions are retained for which $\sum_{i=1}^{p} \lambda_i^{(h)} \geq \text{TNMSE} \sum_{i=1}^{N} \lambda_i^{(h)}$. Thus, the average approximation is roughly the same for all clusters.

   (c) *Global reduction to an average number of dimensions:* the average number of dimensions to retain, p, is provided as input. All the clusters are analyzed together. The products of the eigenvalues $\lambda_i^{(h)}$ times the size of the corresponding clusters $M^{(h)}$ are computed and sorted in ascending order, producing an ordered list of $H \cdot N$ values. Discarding a value $M^{(h)}\lambda_i^{(h)}$ corresponds to removing the ith of the dimensions of the hth cluster, increases the numerator of the NMSE by $M^{(h)}\lambda_i^{(h)}$, and decreases the average number of retained dimensions by $M^{(h)}/M$. The first value in the ordered list is the dimension that has the smallest effect on the NMSE, while the last value has the largest effect. The first ℓ values are discarded, where ℓ is the largest number for which

the average number of retained dimensions is larger than or equal to p. This is the default approach when the number of dimensions is provided as input.

(d) *Global reduction to a target* NMSE: As in the previous case, all the products $M^{(h)}\lambda_i^{(h)}$ of the individual eigenvalues times the number of vectors in the corresponding clusters are computed, and sorted in ascending order. The first ℓ' values are discarded, where ℓ' is the largest number for which the NMSE is less than or equal to TNMSE. This is the default approach when TNMSE is provided as input.

4. *The within-cluster index construction step* operates separately on each cluster $\mathbf{Y}^{(h)}$. In the current implementation, we do not specify a new methodology for this step, rather we rely on any of the known indexing technique. Due to the dimensionality reduction, each cluster is much more amenable to efficient indexing than the entire table. In this paper, the very efficient k-NN method proposed by Kim and Park [18] is used for this purpose, due to its good performance even in medium-high dimensionality spaces (more details are available in [33]).

Sometimes the algorithm can benefit from an initial rotation of the entire dataset into uncorrelated axes (with or without dimensionality reduction), for instance, when the columns of the data table are indeed homogeneously strongly correlated. In this case, an initial application of step 3 should precede the clustering. In other cases dimensionality reduction can conceal cluster structures [11], or the clustering algorithm is insensitive to the orientation of the axes, as in the case of LBG [23] (which is a vector-quantization style clustering method), and no initial transformation of the data should be performed. The index construction algorithm can be applied recursively: Steps 1-3 can be recursively repeated until terminating conditions are met. The resulting indexing structure can be represented as a tree. The data is stored at the leaves, each of which has its own within-cluster index.

The goals of simultaneously minimizing the size of the resulting index and maximizing the speed of the search are somewhat contradictory. Minimizing the size of the index by aggressive clustering and dimensionality reduction is very beneficial when the database is very large. On the other hand, aggressive clustering and dimensionality reduction can also force the search algorithm to visit more clusters, thus reducing the benefits of computing the distances with fewer dimensions.

Since the search is approximate, different index configurations should be compared in terms of recall (Equation (2) and precision (Equation (3)). These two metrics are difficult to incorporate into an optimization procedure, because their values can only be determined experimentally (see Section 4). An optimization algorithm is required to select

the number of clusters and the number of dimensions to be retained for given constraints on precision and recall. This problem can be solved in two steps by noting that both recall and precision are monotonically decreasing functions of the NMSE. Hence an index that minimizes NMSE can be designed for a given data compression objective. If the value of NMSE is large, the optimization with a less stringent data compression objective is repeated. The index is then tested to confirm whether it yields satisfactory precision and recall. This is done by constructing a test set, searching for the k nearest neighbors of its elements using an exact method and the approximate index, thus estimating precision and recall. If the performance is not adequate, the input parameters for designing the index are respecified. Including this last step as part of the optimization is possible, but would have made the initial index design step very lengthy.

A direct consequence of the dimensionality reduction of the index space is the reduction of its index size. The size of the index is affected by the efficiency of data representation and the size of the clustering and SVD information, namely, the $\mathbf{V}^{(h)}$ matrices for coordinate transformation and the number of dimensions retained by the SVD. For large tables and a non-aggressive clustering strategy, the clustering and SVD information are a negligible fraction of the overall data. Thus, we define the *index volume* as the total number of entries preserved by the index. The original volume is then defined as $V_0 \stackrel{\triangle}{=} M \times N$, i.e., it is equal to the number M of vectors in the table times the original number N of dimensions. Given H clusters, denoting with $M^{(h)}$ the number of points and with $p^{(h)}$ the number of retained dimensions in cluster h, the volume after steps 1 and 3 is $V_1 = \sum_{h=1}^{H} M^{(h)} p^{(h)}$. The fraction of volume retained is then simply $F_{vol} = V_1/V_0$ and the mean number of retained dimensions is $N \cdot F_{vol}$. A procedure which determines an "optimal" number of clusters (H) for a given F_{vol} so as to minimize NMSE is described below.

- First note that the reduction in the volume objective directly translates to the mean number of dimensions in the resulting index: $\overline{p}(F_{vol}) = V_1/M$.
- Consider increasing values of H, and for each H use a binary search to determine the value NMSE(H) corresponding to $\overline{p}(F_{vol}) \in [V_1/M - .5, V_1/M - .5]$.
- A point of diminishing returns is reached as H increases, and the search is terminated when NMSE$(H+\delta)$/NMSE$(H) \leq (1+\epsilon)$ (where the parameters ϵ and δ are provided by the user. In our experiments we have used $\epsilon = .01$ and, trivially $\delta = 1$).

In effect, although NMSE^{min} is not achieved exactly by the procedure, selecting the smaller H will result in a smaller index size when the size of the meta-data file is also taken into

account.

The minimum value for NMSE (NMSE^{min}) for the input parameter F_{vol} varies with the dataset. A relatively reasonable F_{vol} (based on experience with similar datasets) may result in an NMSE^{min} which is too large to sustain efficient processing of NN queries. The user can specify a larger value for F_{vol} to ensure a smaller NMSE^{min}. Alternatively, a smaller F_{vol} may be considered if NMSE^{min} is too small.

After the index is created additional tests are required to ensure that the index provides a sufficient precision for a given recall. The query points may correspond to randomly selected vectors from the input table. The index is unsatisfactory if the mean precision $(\overline{P})$ over a test set of samples is smaller than a pre-specified threshold.

3.3 Searching for the Nearest Neighbors

3.3.1 Exact queries

In an exact search all the element of the table that are equal to an $N-$dimensional query point $\mathbf{q}$ must be retrieved. In response to an exact search, $\mathbf{q}$ is first studentized with the same coefficients used for the entire database, thus yielding $\mathbf{q}^{'}$. If an initial rotation and dimensionality reduction was performed before the clustering step 1 during the index construction, the vector $\mathbf{q}^{'}$ is rotated to produce $\tilde{\mathbf{q}^{'}} = \mathbf{q}^{'}\mathbf{V}$, and all the discarded dimensions are set to zero (recall that the dataset is studentized, thus $\boldsymbol{\mu}$ is now the origin of the Euclidean coordinate system). The preprocessed query vector is henceforth denoted by $\hat{\mathbf{q}}$.

The cluster to which $\hat{\mathbf{q}}$ belongs, referred to as the *primary cluster*, is then found. The actual details of this step depend on the selected clustering algorithm. For LBG, which produces a conditionally optimal tessellation of the space given the centroid selection, the index of the primary cluster is $\wp = \text{argmin}_h \left\{ D(\hat{\mathbf{q}}, \boldsymbol{\mu}^{(h)}) \right\}$, (where $D(\cdot, \cdot)$ is the Euclidean distance,) i.e., $\hat{\mathbf{q}}$ belongs to the cluster with the closest centroid. For our implementation of LBG, the computational cost is $O(N \cdot H)$, where again H is the number of clusters. For tree-structured vector quantizers (TSVQ) the labeling step is accomplished by following a path from the root of a tree to one of its leaves, and if the tree is balanced, the expected computational cost is $O(N \cdot \log H)$. Once the primary cluster is determined, $\hat{\mathbf{q}}$ is rotated into the associated reference frame. Using the notation developed in the previous section, we denote by $\mathbf{q}^{'(\wp)} = (\hat{\mathbf{q}} - \boldsymbol{\mu}^{(\wp)})\mathbf{V}^{(\wp)}$ the representation of $\hat{\mathbf{q}}$ in the reference frame associated with cluster $\wp$. Then, the projection step onto the subspace is accomplished

by retaining only the relevant dimensions, thus yielding $\mathbf{q}^{(p)}$. Finally, the within-cluster index is used to retrieve all the records that are equal to $\mathbf{q}^{(p)}$.

By construction, the CSVD index can result in false hits; however, misses are not possible. False hits can then be filtered in a post-processing phase, by comparing $\mathbf{q}$ with the original versions of the retrieved records.

3.3.2 Similarity queries

The emphasis in this paper is on similarity searches, and in particular, on retrieving from the database the k nearest neighbor of the query point $\mathbf{q}$. The first steps of a similarity search consist of the preprocessing described in the exact search context, the identification of the primary cluster, and the projection of the query point onto the associated subspace to yield $\mathbf{q}^{(p)}$. After the query point is projected onto the subspace of the primary cluster, a k-nearest neighbor search is issued using the within-cluster index.

Unlike the exact case, though, the search cannot be limited to just one cluster. If the query point falls near the boundaries of two or more clusters (which is a likely event in high dimensionality spaces), or if k is large, some of the k nearest neighbors of the query point can belong to neighboring clusters. It is clearly undesirable to search all the remaining clusters, since only a few of them are *candidates*, while the others cannot possibly contain any of the neighbors of $\mathbf{q}$. A modified branch-and-bound approach is adopted. In particular, the simple strategy illustrated in Figure 3 is used to determine the candidates. After searching the primary cluster, k points are retrieved. The distance between $\mathbf{q}$ and the farthest retrieved record is an upper bound to the distance between $\mathbf{q}$ and its kth neighbor $\mathbf{n}_k$. A cluster is not a candidate, and thus can be discarded, if all its points are farther away from $\mathbf{q}$ than $\mathbf{n}_k$. To identify non-candidate clusters, the cluster boundaries are approximated by minimum bounding spheres. The distance of the *farthest* point in cluster C_i from its centroid is referred to as its radius, is denoted by R_i, and is computed during the construction of the index.

A cluster is discarded if the distance between its hyper-sphere and $\mathbf{q}$ is greater than $D(\mathbf{q}, \mathbf{n}_k)$, and considered a candidate otherwise (See Figure 3). Note, incidentally, that $\mathbf{q}$ can belong to the hyper-sphere of a cluster (such as, but not necessarily only, the primary cluster), in which case the distance is by definition equal to zero. The distances between $\mathbf{q}$ and the hyper-spheres of the clusters are computed only once. The computational cost is $O(N \cdot H)$, and the multiplicative constant is very small. The candidate cluster (if any) with hyper-sphere closer to $\mathbf{q}$ is visited first. Ties are broken by considering the distances

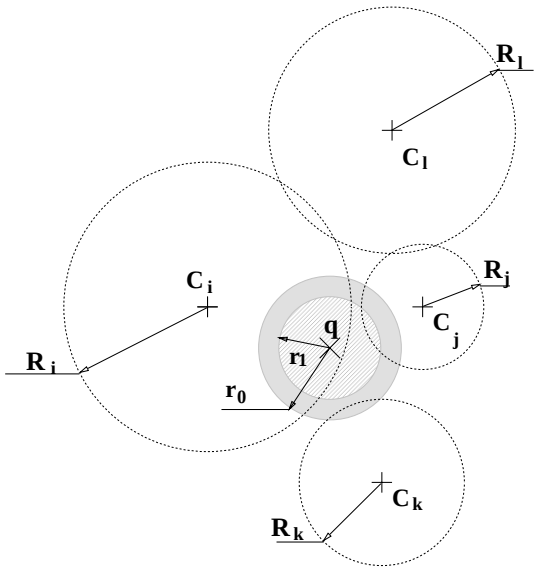

Figure 3: Searching nearest neighbors across multiple clusters. The figure shows three clusters, C_i, C_j, C_k and C_l. The query point is denoted by $\mathbf{q}$. If $D(\mathbf{q}, \mathbf{n}_k) = r_0$, cluster C_i, C_j and C_k are candidates, while C_l is discarded. If, after searching C_i, the updated $D(\mathbf{q}, \mathbf{n}_k)$ becomes equal to r_1, cluster C_k is removed from the candidate set.

between $\mathbf{q}$ and centroids, as is the case, for instance, if $\mathbf{q}$ belongs to the hyper-spheres of several clusters. In the highly unlikely case that further ties occur (i.e., $\mathbf{q}$ is equidistant from the hyper-spheres and the centroids of two clusters), the cluster with smaller index is visited first. If the current NNs list is updated during the within-cluster search, $\mathbf{n}_k$ changes and $D(\mathbf{q}, \mathbf{n}_k)$ is reduced. Thus, the list of non-visited candidates must be updated. The search process terminates when the candidate list is empty.

An alternative, simpler, strategy is outlined below. Denote by $d_{i,j}$ the distance between the centroids of two clusters C_i and C_j, and define the distance between *two clusters* as the distance between their hyper-spheres, i.e., $d_{i,j} - R_i - R_j$. While $d_{i,j} > R_i + R_j$ is true in some cases, $d_{i,j} < R_i + R_j$ is also possible. It is even possible for the K-means method to generate one cluster embedded inside another [20]. During the search phase, the clusters that intersect the primary cluster are checked in increasing order of distance, while clusters with positive distances from the target cluster are not checked. This simple strategy has the advantage of being static (since $d_{i,j}$ are compute at index-construction time), but does not guarantee that all the relevant clusters are visited.

Finally, more complex approaches can rely on the approximation of the cluster boundaries by more complex surfaces, such as, for instance, hyper-ellipsoids aligned with the eigenvalues of the cluster. Using better approximations of the cluster boundaries reduces the possibility of visiting non-candidate clusters, however it can significantly complicate the identification of the candidates. The outlined strategies can also be used as successive filtering steps: the static approach can be used first, the identified candidates can be filtered using the hyper-sphere method and the surviving clusters can be further pruned with more advanced strategies. As the benefits of the different methods depend on the database, the effectiveness of the actual implementation of the different strategies and of the within-cluster indexes, the best approach can only be determined empirically.

Care must be taken when performing the within-cluster search. In fact, during this step the within-cluster algorithm computes the distances between the projections of $\mathbf{q}$ and of the cluster points onto a subspace (hyperplane) of the original space. Simple geometry shows that two points that are arbitrarily far apart in the original space can have arbitrarily close projections. In particular, while the points within the cluster are by construction, on the average, close to the subspace, the distance between the query point and the subspace can be large, especially when clusters other than the primary one are searched. The search algorithm must therefore account for this approximation by relying on geometric properties of the space and of the index construction method. As shown in Figure 4 for a 2-dimensional table, the Euclidean distance D_p between the query point $\mathbf{q}$ and a point $\mathbf{p}$ on the subspace (in the figure, the centroid), can be computed

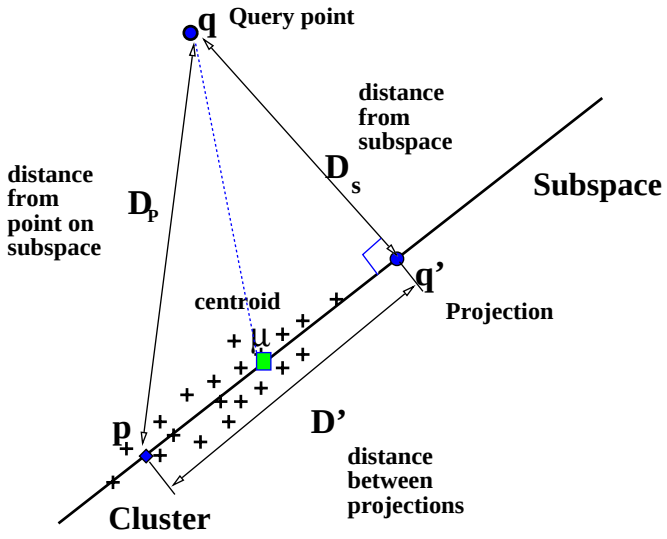

Figure 4: Decomposing a template into its projection onto the subspace of a cluster and its orthogonal component.

using the Pythagorean theorem, as $D_P^2 = D_s^2 + D^{'2}$, where D_s is the distance between $\mathbf{q}$ and the subspace, and $D^{'}$ is the distance between the projection $\mathbf{q}^{'}$ of $\mathbf{q}$ and $\mathbf{p}$. Thus, during the within-cluster search, the quantities $\left(D^{'2} + D_s^2\right)$ are substituted for $D^{'2}$. To compute D_s^2, the Pythagorean theorem is invoked again: the cluster centroids $\boldsymbol{\mu}^{(i)}$ are known (in the original space), thus $D(\mathbf{q}, \boldsymbol{\mu}^{(i)})$ are easily computed. The rotation of the space corresponding to $\mathbf{V}^{(i)}$ does not affect the Euclidean distance between vectors, thus the distance between $\mathbf{q}$ and the subspace is just the distance between $\tilde{\mathbf{q}}$ and the projection $\tilde{\mathbf{q}}^{'}$. This quantity can be easily computed during the dimensionality reduction step, by squaring the values of the discarded coordinates, adding them and taking the square root of the result.

3.4 Within Cluster Search

If within-cluster search relies on sequential scan, the long-term speedup yielded by CSVD over sequential scan of the entire database is at most equal to the ratio of the database size to the expected number of samples in the visited clusters, times the compression ratio $1/F_{vol}$. Thus, if CSVD reduces the dimensionality of the database by a factor of 5 and if on average the visited clusters contain 10% of the database, CSVD is at most 50 times faster

than linear scan. This figure does not account for the costs of computing the distances between search vectors and clusters, of projecting the search vectors, of computing the correction factors, and of retrieving a larger number of neighbors to achieve a desired recall. Hence, the observed speedup under the above assumptions is smaller.

Further benefits can be obtained from a careful selection of an indexing method for within-cluster search. We start by making a simple observation. Let F be a distribution over $\mathbb{R}^N$ having diagonal covariance matrix, with variances satisfying $\sigma_i^2 \geq \sigma_j^2$ for each $i < j$. Thus, the variance of F along the 1st dimension, σ_1^2, is the largest, and the variance along the last dimensions, σ_N^2 is the smallest. Let $X^{(1)}$ and $X^{(2)}$ be two independent samples drawn from F; their squared Euclidean distance can be written as $D^2 = \sum_{i=1}^{N} (X_i^{(1)} - X_i^{(2)})^2$, the expected value of which is $\mathbb{E}D^2 = 2\sum_{i=1}^{N} \sigma_i^2$. Thus, the 1st dimension contributes the most to $\mathbb{E}D^2$, followed by the second dimension, and so forth. Recall that SVD produces uncorrelated features, ordered by decreasing variance, and therefore the distribution of the projected vectors in each cluster satisfy the above assumptions.

An indexing scheme that partitions the space into hyperrectangles aligned with the coordinate axes, and prunes the search space using first the first coordinate, then the second, and so on, clearly benefits from the discussed property. Among the many existing schemes of this family, we have adopted the one proposed by Kim and Park [18], based on the ordered partition.

The ordered partition is a tree-based index for nearest neighbor search, that recursively divides the space along a different dimension. During step n of the index construction, the current dataset is divided into a predefined number of equal-size groups, by appropriately partitioning the nth coordinate of the space. Each group is then recursively divided independently of all the others. The resulting tree is in general balanced and has terminal nodes corresponding to hyperrectangles in $\mathbb{R}^N$, and containing roughly the same number of samples. In the original paper, the number of splits along each coordinate was a function of the total number of samples and of the number of dimensions. An elegant search technique was also proposed. First, the terminal node containing the query sample is identified and exhaustively searched. A list of the current best k matches is constructed and maintained. When a node is completely searched, a backtracking step is taken, consisting of identifying all the siblings of the current node that could contain one or more of the k nearest neighbors of the query sample. If it exists, the best candidate sibling is visited. Otherwise the search continues from the previous level of the tree, until all the candidate children of the root have been searched.

Kim and Park's index is very efficient in low-dimensional spaces. However, as the di-

mensionality increases, it suffers from the same problems of all the other indexing schemes mentioned in the introduction. In particular, it is easily seen that, to grow a depth-N tree, the database must contain at least 2^N samples. Also, searching an ordered partition based on a single split per dimension results in visiting a large part of the tree during the search process. Thus, in 20 or more dimensions the index is usually inefficient. However, the last statement is not true if the first few dimensions of the search space account for a large part of the data variability, and if the ordered partition indexes only those dimensions.

When constructing the index for individual clusters, where the rotated dimensions are ordered by their variance, we limit the depth of the ordered partition tree. Often, the first few dimensions of the rotated space account for a large part of the overall variability, and the index offers significant advantages over sequential scan.

Then, during the construction of the index, we fix the size of the leaves, and impose a minimum fan-out for the non-terminal nodes. The choice of the size of the leaves depends on how the index is managed. If the index resides on disk and portions of it are retrieved to main memory accessed, then the search is I/O bound. In this case, the number of samples in each leaf depends on the size of the physical disk page and on the number of bytes required to represent each sample. If the database resides in main memory, as it is becoming increasingly common with the advent of very large main memories, the search is CPU-bound (namely, it is determined by the CPU/caches/main-memory complex,) and the size of the terminal nodes is chosen to best utilize the memory hierarchy of the server architecture during sequential scan.

4 Performance Study

4.1 Materials and Methods

Before proceeding with the main discussion, we provide some background material about the numerical packages and the input data used in our study.

SVD and eigenvalue analysis is a basic ingredient of numerical packages e.g., [26]. In fact the computational routine for SVD can be used to obtain the eigenvalues and eigenvectors of the covariance matrix $\mathbf{C}$. Experiments with tables of texture features revealed that $\lambda_i \approx s_i^2/M$ with very high accuracy, except for the smallest of the eigenvalues, which are anyway irrelevant to CSVD (see Section 2.2). When the table $\mathbf{X}$ is too large to fit in main memory due to a high value for M, the covariance matrix can be computed in

a single pass over $\mathbf{X}$ from disk [19]. The computation and writing of the appropriate columns of the dimensionality reduced matrix $\mathbf{Y}$ will require another pass over $\mathbf{X}$.

The experiments have been performed on several datasets. The results reported were based on three tables of texture features obtained with a Gabor decomposition [2]. The small table, containing 16,129 rows and 60 columns, and the large table, containing 160,000 rows and 55 columns, were extracted from a small database containing 37 satellite images of selected regions in Alaska of size 512×512 pixels each, acquired with a Synthetic Aperture Radar. The mid-size table, containing 56,688 rows and 60 columns, was extracted from the central 1000×1000 pixels of band 3 of four Landsat MSS images from different parts of the country.

The implementations of clustering methods reported in this paper can handle very large datasets, as long as they can be processed efficiently by the virtual memory system (thousands of vectors and tens of columns requiring several hundred megabytes). Recently proposed methods for clustering disk resident data (see [19] for citations), have been used with two dimensional tables, but are applicable to the problem at hand with appropriate extensions. When applicable, initial dimensionality reduction of the input dataset via SVD and dimensionality reduction provides a partial solution of the memory residence problem, by reducing the size of the input to the clustering algorithm. This step was not performed during the experiments. The reported results were obtained with a standard LBG clustering algorithm, where the seeds were produced by a tree structured vector quantizer.

As the construction of the index always includes a randomization step (since the iterative procedure used to construct the quantizer uses a set of randomly selected seeds), the measurement corresponding to each data point in the graphs represents the average over 100 different runs of index construction. All the measures of quantities related to queries (precision, recall, timing etc.) are, unless otherwise stated, the average of 5000 queries per each construction of the index, where the query templates were constructed by uniformly sampling the database and adding a small randomly generated vector. When comparisons were made with sequential scan, the same query examples were used for both sequential scan and CSVD.

Experiments were run on a single processor of an IBM RS/6000 43P workstation model 260, with 2 gigabytes of main memory and 4Mb of second-level cache per processor. The processors are 200-MHz POWER3 64-bit RISC processors, with 2 floating point units (supporting single-issue MultiplyAdd operations), 2 load-store units, 3 integer units and 1 branch-dispatch unit. Each processor can complete up to 4 instructions per cycle, (one per each group of units), and support fast division and square roots. No additional workload

was concurrently run on the second processor. Running times were collected within the applications by means of system calls.

When measuring average speedup, we measured the time to complete a large number of queries and divided it by the number of queries, rather than computing a per-query speedup and averaging over a large number of queries. The selected procedure better captures the behavior of the system under a sustained workload, the other one is infeasible due to the coarse granularity of the system timer.

4.2 The Tradeoff between Index Volume and NMSE

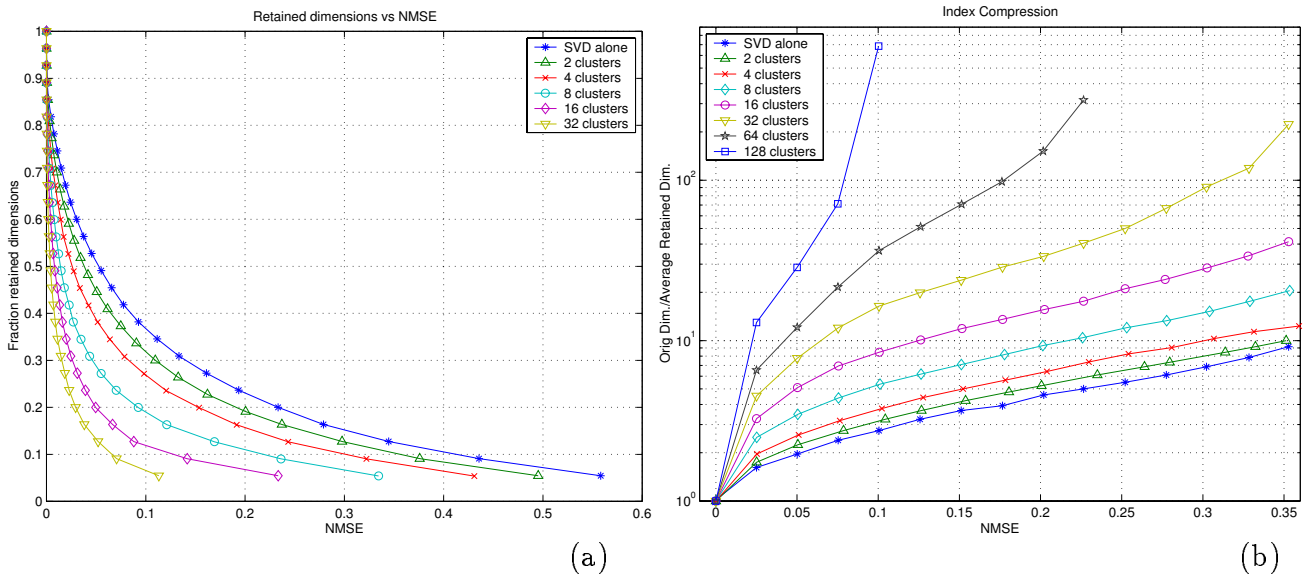

Figure 5: (a) Relation between retained volume and Normalized Mean Squared Error. The plots are parameterized by the number of clusters. (b) Database compression as a function of the NMSE, parameterized by the number of clusters. The cost of storing the centroids and the dimensionality reduction information is ignored. Compression ratios of more than 60 are possible when all the dimensions of several clusters are discarded, in which case the points are represented by the corresponding centroids. Table size: $16,129 \times 60$.

The relationship between the approximation induced by CSVD, measured in terms of NMSE, and the number of retained dimensions (p), or equivalently the fraction of retained volume (F_{vol}), was discussed in Section 2.2. Since SVD, and hence CSVD, is only desirable

if NMSE is small for high data compression ratios, we have quantified experimentally this relation.

In Figure 5(a), the percentage retained volume F_{vol} is plotted as a function of the NMSE. As the relation between F_{vol} and NMSE is a function of the number of clusters H, the plot shows different curves parameterized by different values of H. The line corresponding to a single cluster shows the effects of using SVD alone without clustering. There is a significant drop in F_{vol} for even small values of NMSE, suggesting that few dimensions of the transformed features capture a significant portion of the variance. However, the presence of local structure is quite evident: when the NMSE is equal to .1, using 32 clusters reduces the dimensionality of the search space from 60 to less than 5, while simple SVD retains 24 dimensions.

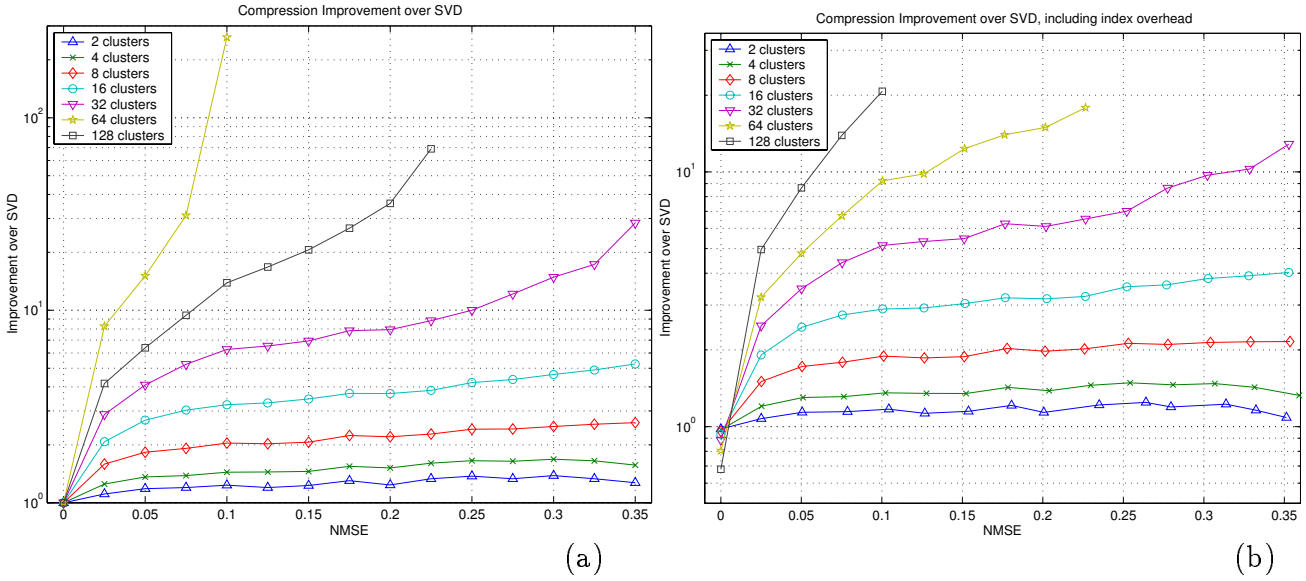

Figure 6: (a) Improvement in dimensionality reduction (compression) over simple SVD as a function of the NMSE. (b) Improvement in index size including the clustering and dimensionality reduction overhead. Both figures show plots parameterized by the number of clusters. Table size: $16,129 \times 60$.

The reduction in the fraction of retained volume F_{vol} as the NMSE grows, is an increasing function of the number of clusters. For example, NMSE$=.05$ results in $F_{vol} \approx .52$ for $H = 1$ cluster, and $F_{vol} \approx .12$ for $H = 32$ clusters, indicating that CSVD outperforms SVD to a significant extent. This confirms the presence of local structure that cannot be captured by a global method such as SVD.

Figure 5(b) shows the overall compression ratio achieved as a function of the NMSE, parameterized by the number of clusters, computed as the ratio of the number of dimensions in the search space to the average number of retained dimensions required to attain the NMSE objective. The approach number 3d to dimensionality reduction was used to produce the plot. Figure 6(a) shows the improvement in compression ratio over simple SVD, as a function of NMSE. As before, the larger the number of clusters, the better CSVD adapts to the local structure of the database, and the higher is the compression ratio that can be achieved for the same NMSE of the reduced dimensionality index to the original database. Note that significant advantages in compression ratio over simple SVD are attained in the interesting region $.05 \leq NMSE \leq .15$. If the overhead due to the index is accounted for, the compression improvement over simple SVD is smaller, as seen in Figure 6(b). This overhead is equal to the cost of storing, for each cluster, both the centroid and the part of the matrix V required to project the samples onto the corresponding subspace. The first component is equal to $H \cdot N$ (the number of clusters times the number of dimensions of the original space), and the second is equal to the sum over all clusters of N times the number of retained dimensions. The effect of the overhead is very substantial given the small database size, but are much less pronounced in experiments based on the larger databases. While eventually the trend seen in Figure 5 is reversed, as the cost of adding an additional cluster is larger than the saving in dimensionality reduction, this effect is moderate for the parameter range used in the experiments, even when the small database is used. The overhead is small for $H \leq 16$, where the overall compression ratio is reduced by a factor of 1.2 or less, and is important only for $H \geq 32$. For $H = 32$ the overall compression ratio is reduced by factor of 1.1 at low NMSE, and 1.7 at high NMSE.

A different view of the same data is provided in Figure 7(a), where the fidelity index (defined as $1-$NMSE) is plotted as a function of number of clusters and is parameterized by the percentage of retained volume. Figure 7(b) plots the ratio $NMSE_{SVD}/NMSE_{CSVD}(H)$, which measure the increase in fidelity to the original data, parameterized by the percentage of retained volume. Note that CSVD always outperforms SVD, and that there is a substantial reduction in NMSE as the number of clusters H, for all values of F_{vol} considered. Similar results are obtained with even larger datasets, with over 200,000 rows and 42 dimensions, but the optimum value for H depends on the dataset, F_{vol}, and the quality of the clustering method.

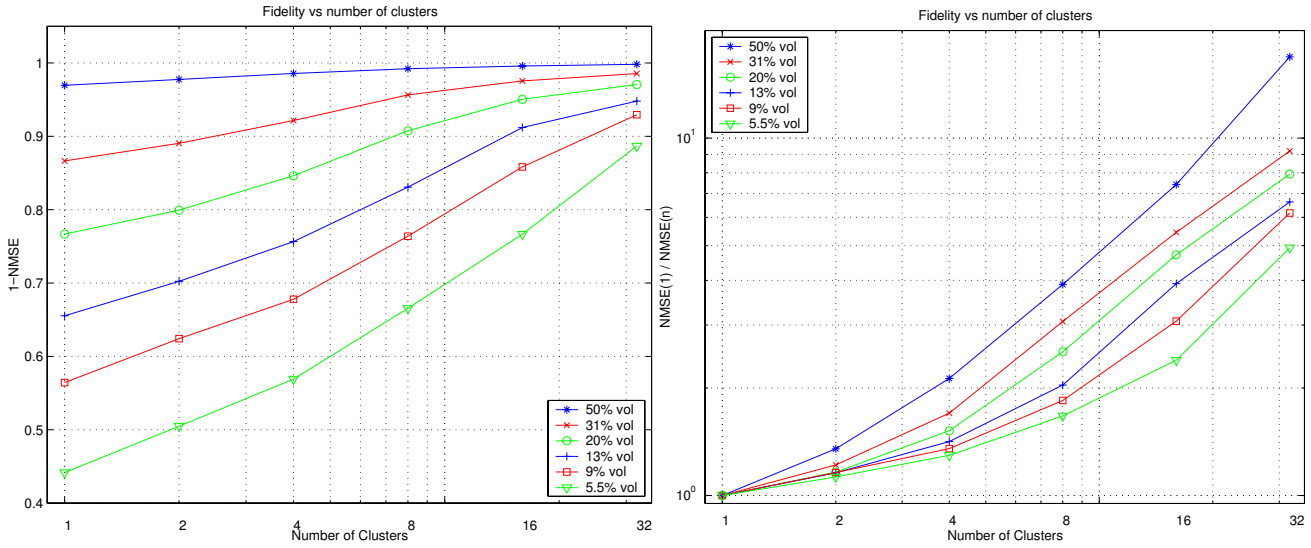

Figure 7: (a) Dependence of fidelity (defined as 1−NMSE) on the number of clusters, parameterized by the volume compression. (b) Increase in fidelity (defined as the inverse ratio of the NMSE's) over simple SVD, parameterized by the volume compression (log-log plot). Table size: $16,129 \times 60$.

4.3 The Effect of Approximate Search on Precision and Recall

The price paid for improving performance via dimensionality reduction is that the resulting search becomes approximate. Here, the source of the approximation is the projection of the vectors in the database onto the subspace of the corresponding cluster: the retrieval phase is based on the (exact) distances between the query template and the projected vectors rather than the original vectors. We define the retrieval to be exact if the approximation does not change the ordering of the distances between the returned records and the query point, i.e., the original ranking of points is preserved, which is a common occurrence when the discarded variance is negligible. However, when the approximations due to the projection of the database records onto the subspaces of the corresponding clusters yield larger errors, discrepancies in distance ordering occur, giving rise to erroneous rankings. Consequently, when issuing a k-nearest-neighbor query, some undesired records are retrieved and some desired records are discarded. To quantify these effects, we can use the *recall* parameter, defined in Equation (2). The recalls of individual queries are not a

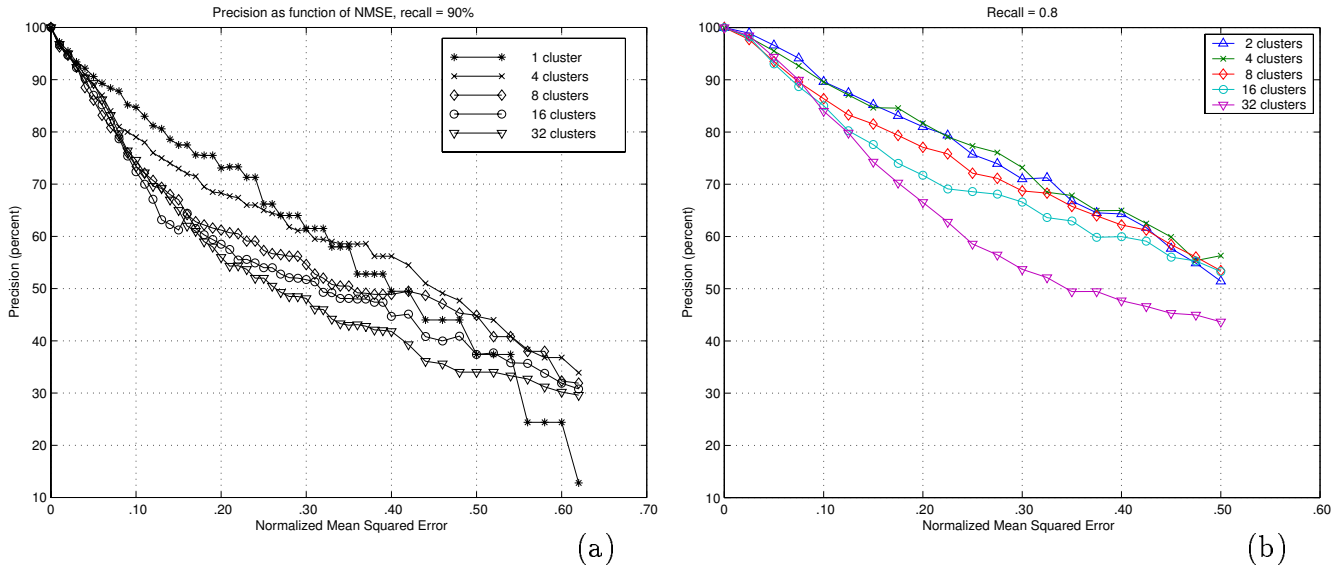

Figure 8: Precision as function of NMSE, for different number of clusters, for recall = .9 (a) and recall = .8 (b). Table size: $160,000 \times 55$.

good measure of the accuracy of the algorithm. Worst-case recall is also an inappropriate measure for the problem at hand, as the clustering step was selected to minimize the average distance, rather than the maximum distance, between points and corresponding centroids. If this were the goal, appropriate modification to the index construction and the search strategy would have to be made.

Thus, in the following, recall will be interpreted as "expected recall", and estimated using empirical averages. Since the retrieval is known to be approximate, the number of results specified to the algorithm should be larger than the number of desired results, to attain a pre-specified recall level. For example, the user could specify a desired recall of $R \geq .9$ and the algorithm would retrieve, say $n = 30$, vectors, of which $k' = 19$ belong to the $k = 20$ nearest neighbor. The efficiency of the retrieval, measured in term of *precision* (Equation (3)) is then $P = k'/n = .63$, and the recall is $R = k'/k = .95$

For a given database, CSVD precision and recall are a functions of the number of clusters, of the desired number of neighbors, and of the data compression. Note that data compression can be specified either in terms of a desired average number of retained dimensions or in terms of a fidelity criterion, i.e., NMSE. Here we present results summarizing the dependency of precision on the discarded variance, and on the retained volume, for a fixed recall level and number of neighbors. Each plot contains a family of curves

parameterized by the number of clusters.

To estimate the value of P for a specified recall value ($R_{threshold}$) and a pre-defined number of neighbors k, we determine for each query point $\mathbf{q}$ the minimum value of n, $n_{\mathbf{q}}$, yielding $k' \geq k * R_{threshold}$ correct results. Then $P(\mathbf{q}) = k'/n_{\mathbf{q}}$, and the expected precision is estimated as the average of $P(\mathbf{q})$ over a set of query points of the same size as the database. Figure 8 characterizes the mean precision ($\overline{P}$) for $k = 20$ as a function of the NMSE for various number of clusters, and for two values of recall, $R_{threshold} = .9$, and $R_{threshold} = .8$ Note first that $\overline{P} > .5$ for NMSE up to .4 and $R = .9$, and up to

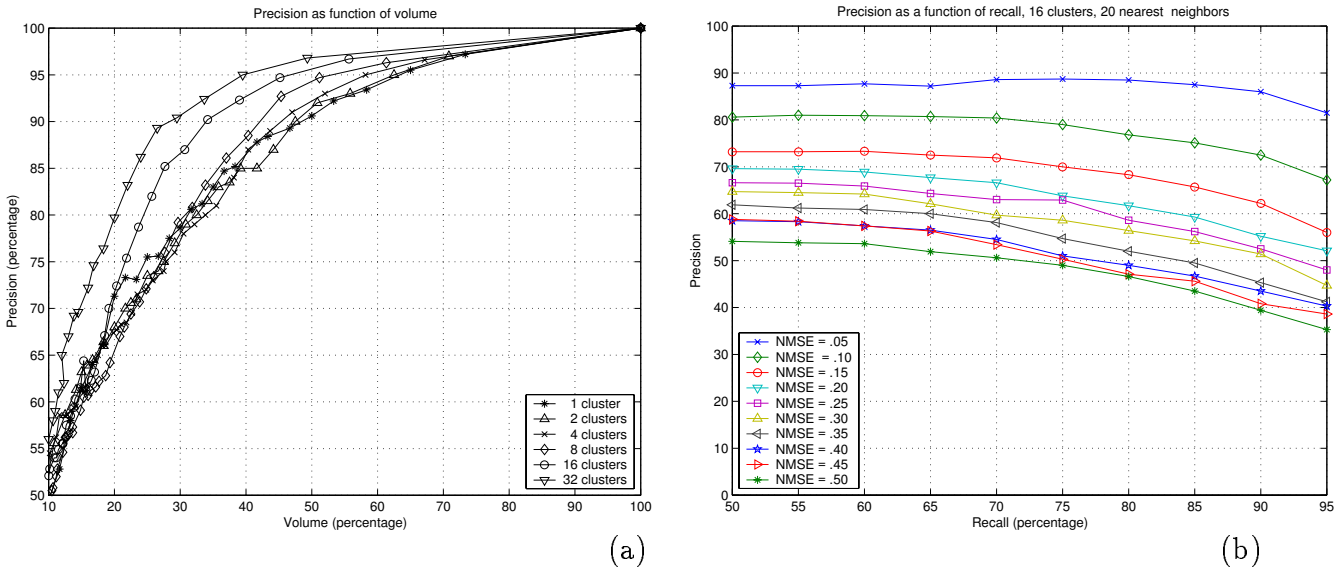

Figure 9: Precision as function of the retained index volume (a), for different number of clusters. Recall = .9, table size $160,000 \times 55$. Precision as a function of recall (b). Table size = $16,129 \times 60$, number of clusters = 16, desired number of nearest neighbors = 20. The curves are parameterized by the retained variance.

NMSE = .5 and $R = .8$. A single cluster provides the highest precision when NMSE is small. This can be attributed to the inaccuracies in reconciling inter-cluster distances and results in reduced precision (see Figure 4). Unsurprisingly, clustering reduces precision for a given NMSE. The higher the clustering degree, the higher is the average dimensionality reduction. Projections of points within individual clusters are much closer than they would be if simple SVD were used, and therefore some discrimination is lost. The effect is moderate for $NMSE < .10$, and becomes more substantial afterwards. In some occasion we have noticed that, beyond this threshold, entire small clusters are projected onto their

centroid. Eventually, however, the trend is reversed, and for $NMSE > .5$ the precision difference between CSVD and simple SVD becomes smaller. Also, in the interesting zone NMSE $< 10\%$, good precision values are observed across the board. The reported results are typical of those obtained on a large number of combinations of the values of k and $R_{threshold}$.

Figure 9(a) shows the mean precision as a function of F_{vol} parameterized by the number of clusters, with recall fixed at .9. Note that $H = 32$ clusters result in the highest precision curve, and that for the same volume reduction level the best information is preserved with a higher number of clusters. This was to be expected, since for fixed index compression the NMSE decreases significantly with the number of clusters (Figure 7), which results in a higher precision. Finally, it is interesting to note that for 32 cluster, even when the data compression ratio is 5:1, the precision is around 80%, and when the compression ratios is 8:1 the precision remains slightly less than 70%. In this last case, to achieve $R = .9$ during an k-NN search, it is enough to retrieve $1.5\ k$ results and post-process them to rank the results correctly.

In Figure 9(b), the precision versus recall curve parameterized by different values of NMSE. The experiment was carried out with sixteen clusters, and similar graphs can be obtained for different number of clusters. It is observed that the NMSE has the primary effect on precision and the effect of increased recall is secondary. Precision deteriorates with increasing recall, but the drop is relatively small.

In an operational system, the empirical precision versus recall curve $\overline{P}(R)$ would be stored with the indexing structure. The specification of a query consists then of providing a query point, the minimum desired expected recall $R_{threshold}$ and the desired number of nearest neighbors k. The system would then retrieve $n = k/\overline{P}(R_{threshold})$ vectors using CSVD, score them in the original search space, and return the top k matches. This procedure yields on average the desired precision.

Alternatively, a more conservative approach would be to construct a histogram of precision values parameterized by recall, and use a precision quantile smaller than the average, $n = k/P_q$, which would yield the desired recall $1 - p$ of the times.

4.4 Retrieval Speedup

The main purpose of an indexing structure is to increase the retrieval speed over a linear scan of the entire database. As reducing the number of dimensions makes it possible to store the index to the entire databases in the main memory of a computer (current

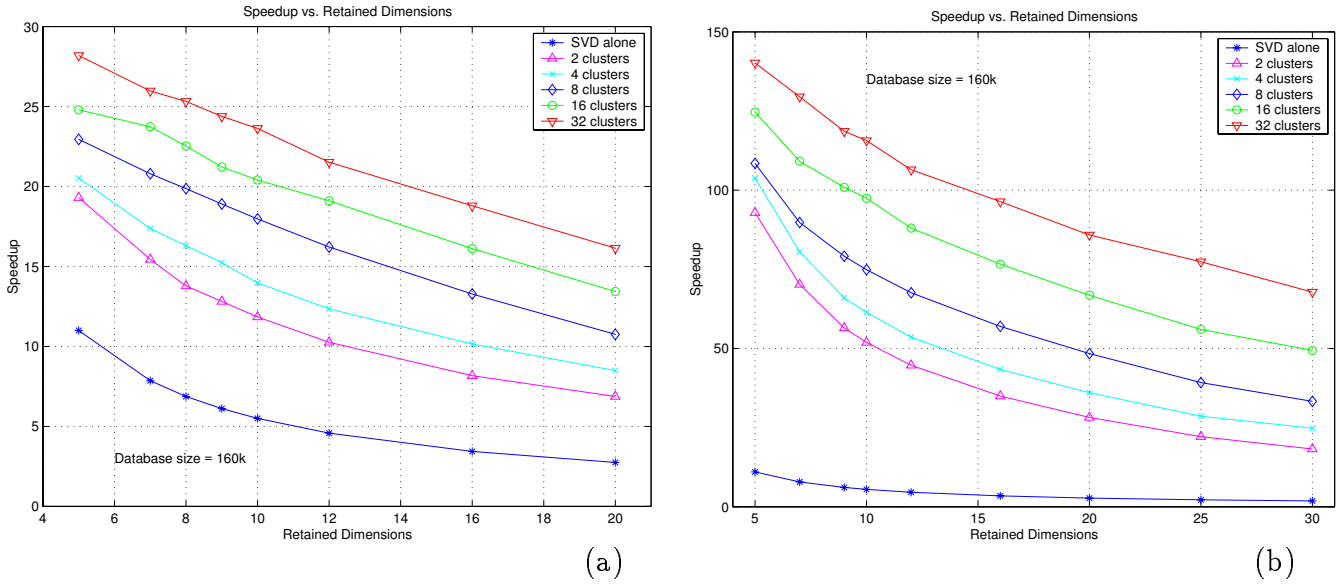

Figure 10: Speedup as a function of retained dimensions for texture feature databases containing (a) 16,129 60-dimensional entries and (b) 160,000 55-dimensional entries. Note that here the line corresponding to 1 cluster depicts the effects of simple SVD+dimensionality reduction, where the maximum observed speedup is about 10.

small and mid-range database servers have gigabytes of primary store, and very large memories are becoming increasingly pervasive), it is important to quantify the in-memory performance of the index. The comparison is performed with respect to sequential scan, rather than with respect to other indexes. As it is relatively simple to implement very efficient versions of sequential scan using highly optimized native routines, it can be universally used as baseline for every indexing method. On the contrary, to perform a fair comparison, competing indexing structures would have to be recoded and optimized for the specific architecture on which the test is run.

Figure 10 shows the behavior of the speedup as a function of the number of retained dimensions. The search was for the 20 nearest neighbors of the query template. The number of actually retrieved neighbors was selected to ensure a minimum average recall of .9, and therefore varies with the number of clusters and the number of retained dimensions. Note that, for indexes with 32 clusters, the observed speedups in the smaller database ranged between 15 and 30, while for the larger database the observed speedups are five times larger, thus showing that, within the range of database sizes used, the index scales with the number of samples in the table.

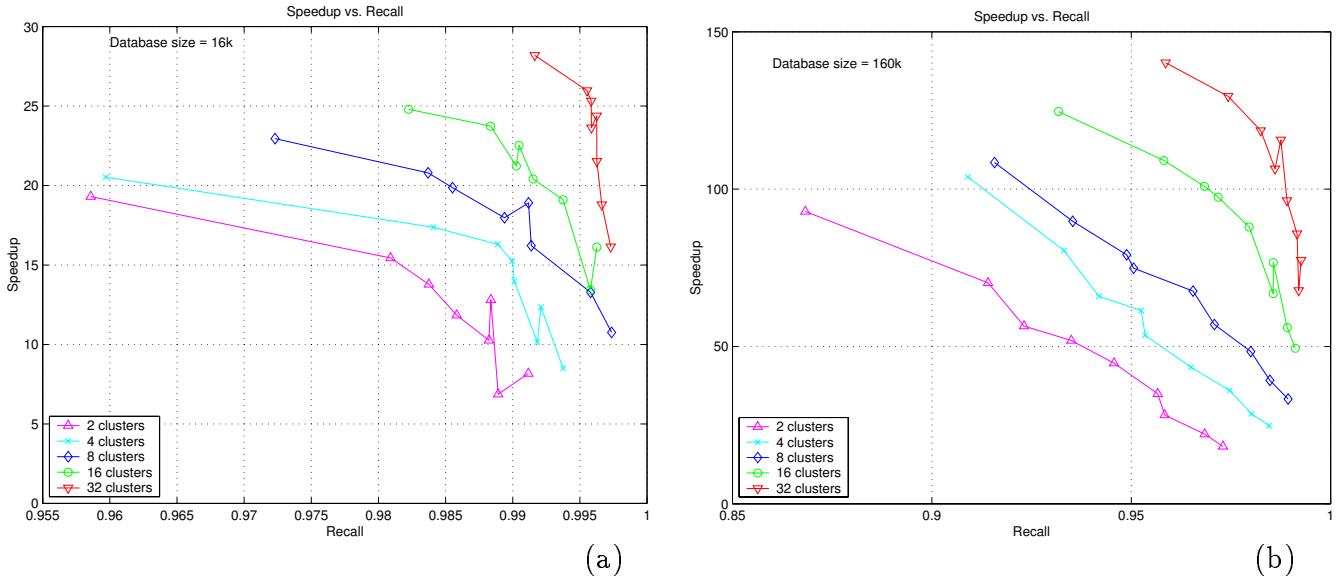

Figure 11: Speedup as a function of recall, for texture feature databases containing (a) 16,129 60-dimensional entries and (b) 160,000 55-dimensional entries.

A different view of the data is obtained by plotting the speedup as a function of the average recall, as shown in Figure 11. Note how in the 32 cluster case, and for the larger database, speedups of 140 are observed while maintaining 96% recall on a 20 nearest neighbor search, which implies that on average 19 of the 20 desired results were retrieved. The recall can be improved to 97.5% while still achieving 130 fold speedup: here more than half of the 100,000 queries corresponding to each point in the graph returned all the desired 20 nearest neighbors. It is also interesting to note how in this case the fine structure of the data is better suited to a larger number of clusters.

4.5 CSVD and within-cluster indexing

As discussed in Section 3.4, the increase in search speed is due to both CSVD and its effects on the within-cluster search index. In this section, we quantify the contributions of these effects, using experiments based on the larger of the data sets.

In the first set of experiments, the test query returns the 20 nearest neighbors of the template. The recall level is fixed to be between .95 and .96, thus, on average, at least 19 of the 20 results are correct. To obtain the desired recall, the number of actually retrieved

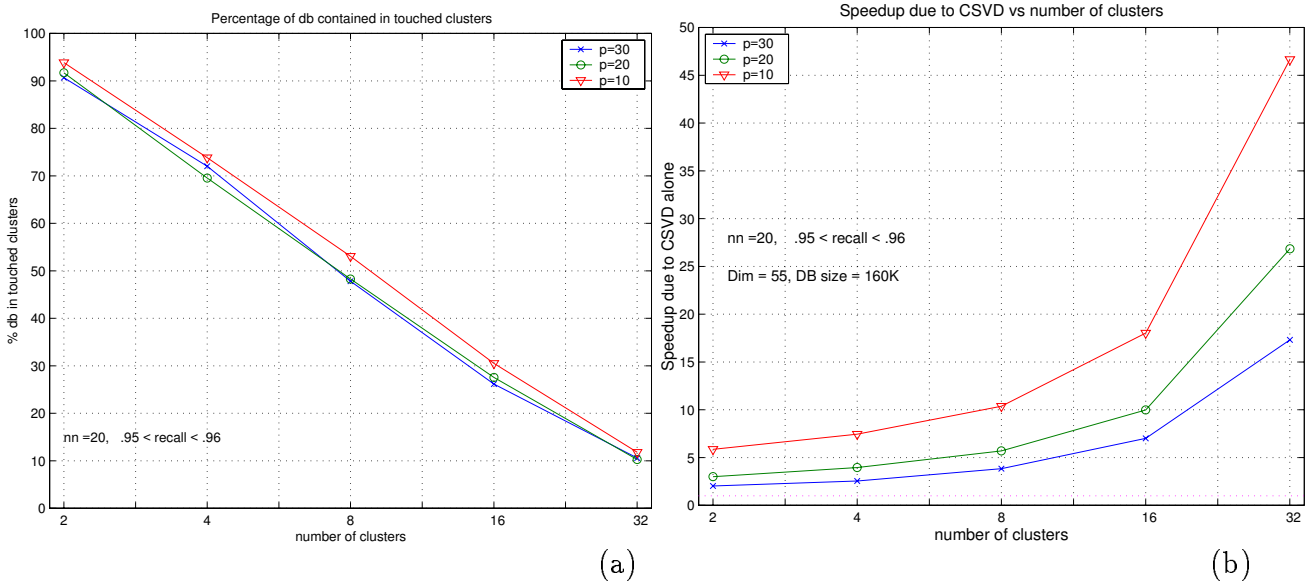

Figure 12: (a) The percentage of the database in the touched clusters. (b) The increase in retrieval speed (b) due to CSVD alone. Both quantities plotted as a function of the number of retained dimensions and of the number of clusters. Table size $=56,688 \times 60$.

samples is selected to yield the correct point of the precision-vs-recall curve (Figure 9(b)) computed for the desired number of clusters. Additional diagnostic code is enabled, which records the average number of visited clusters, the average number of visited terminal nodes and the average number of non-terminal visited nodes. The diagnostic code has a side effect of slightly reducing the observed speedup, thus the results shown in this section are slightly worse than those shown in previous sections. Each data point corresponds to the average of 80,000 retrievals (10,000 retrievals for each of 8 indexes).

Recall that the average increase in retrieval speed due to CSVD alone (i.e., when the within-cluster search is exhaustive) is roughly equal to the index compression ratio times the expected ratio of the database size to the number of elements contained in the clusters visited during the search. Figure 12(a) shows the average percentage of the database contained in the visited clusters, as a function of the overall number of clusters in the index, and parameterized by the number of retained dimensions. The number of clusters in the index has the largest effect on the percentage of the database visited during queries. We have observed that, when the index contains 2 clusters, both are searched during more than 90% of the queries; when the index contains 8 clusters, slightly less than half of them are visited on average, while when the index contains 32 clusters, only

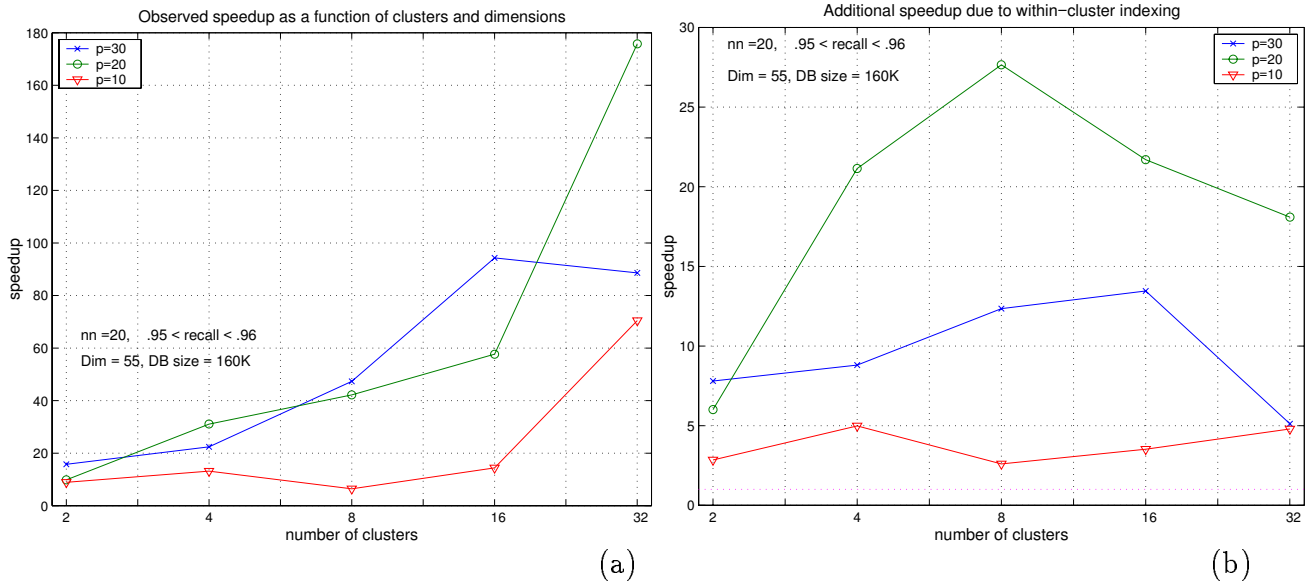

Figure 13: (a) The observed speedup; (b) the additional speedup due to within-cluster indexing; both quantities plotted as a function of the number of retained dimensions p and of the number of clusters. Table size $=56,688 \times 60$.

three are searched on average. An important consequence is the following: If the cluster contents were paged and read from disk during query processing, using SVD would require loading the entire database each time. When the index contains 32 clusters, on average only 10% of the database would be read from tertiary storage in response to a query. From the figure, we also note that the effect of the number of retained dimensions is secondary: in CSVD the selection of the primary cluster and of the candidates is performed in the original data space, and it is therefore not significantly affected by the within-cluster dimensionality reduction. However, the dimensionality reduction has a direct effect on the precision-recall curve (Figure 9(b)), and, to achieve the desired 95% recall, the number of retrieved samples increases with the number of discarded dimensions, thus resulting in a slightly larger average number of visited clusters and a slightly higher curve for smaller number of retained dimensions.

Figure 12(b) displays the expected speedup over linear scan under the assumption that each of the visited clusters is exhaustively searched. In the figure, the additional costs of projecting the query vector, of identifying the primary and candidate clusters, and of maintaining the list of partial results, are ignored. Here, the dimensionality reduction plays a major role, as it controls the data volume to be read from memory to the processor and

the number of floating point operations required to compute distances.

Figure 13(a) shows the actually observed speedup when using within-cluster indexing, which also includes the costs of searching CSVD and of maintaining the list of partial results. The data points in the figures were obtained with the Kim-Park index, by requiring a minimum fan-out of 4 for internal nodes, and a minimum number of points per leaf equal to 4. These observed values are larger than the corresponding ones in Figure 12(b) by the factor depicted in Figure 13(b). Recall that, for generic high-dimensional feature space, we would expect the within-cluster index to have some moderate effect in 10 dimensions, but to be essentially useless in 20 or more dimensions. The experimental evidence, however, contradicts this intuition. When 30 or 10 dimensions are retained, the dependence on the number of clusters is moderate, and the effect of the indexing is an increase in speed of around 10 and 3 respectively. In 20 dimensions, the increase in speed is rather dramatic, ranging from 18 to 28 times for $H > 2$.

We conclude that CSVD transforms the data in a form that is well suited for use with some multidimensional indexing, in particular with recursive partitioning methods that split the space using hyperplanes perpendicular to the coordinate axes, that split across one coordinate at a time, and that can either select the dimensions for partitioning or that partition them in a round robin fashion.

4.6 Within-cluster indexing design

The within-cluster search cost can be divided into two components: walking the tree to identify candidate leaves, and exhaustively searching the leaves. We include the cost of maintaining the list of current candidates in the first component.

When the entire index is in main memory, the cost of exhaustively searching leaves is essentially determined by the CPU, provided that the implementation of the distance computation algorithm makes good use of the memory hierarchy. Figure 14(a) shows the relative computational costs of visiting the tree and the leaf nodes recorded in the experiments used to construct Figures 12 and 13. It is immediately clear that decreasing the number of retained dimensions increases the cost of visiting the tree. This is not surprising: the cost of exhaustively searching a leaf is directly proportional to the length of each contained vector. If this were the only effect, the relative cost of searching the leaves in 20 and 30 dimensions would be less than twice and three times, respectively, the cost of searching the leaves in 10 dimensions. The figure, however, suggests the presence of secondary effects, which increase these relative costs. The Kim-Park tree is used to

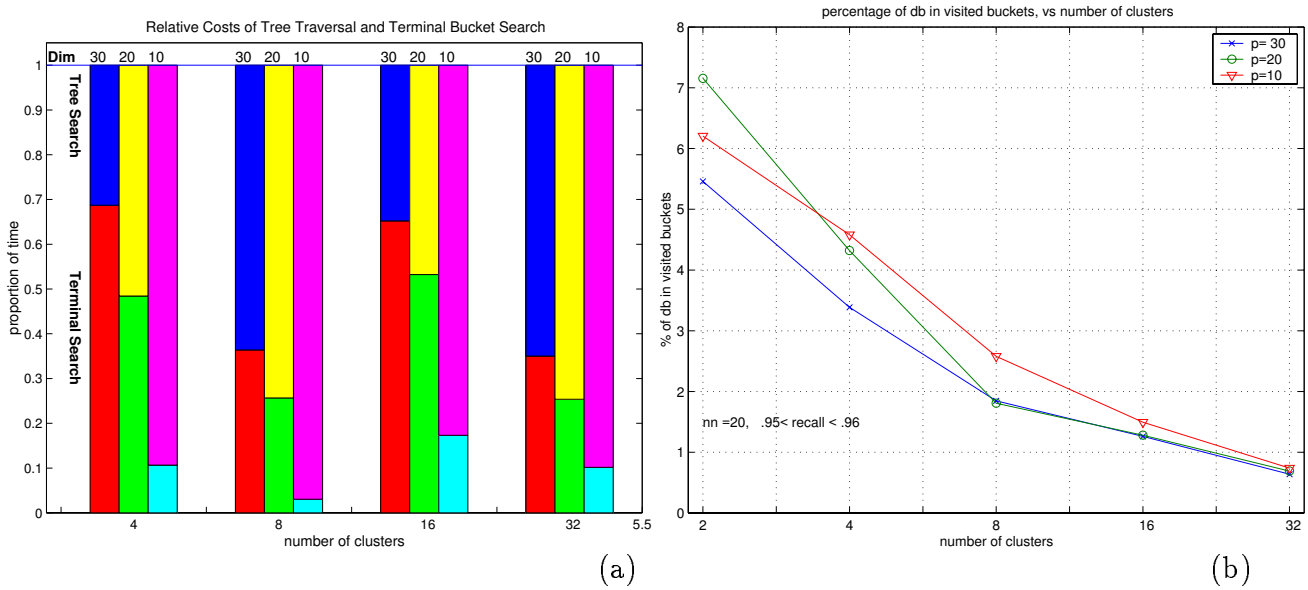

Figure 14: (a) Relative costs of tree and terminal bucket search. (b) Average percentage of the database in the exhaustively searched terminal bins, parameterized by the number of retained dimensions p. Table size $=160,000 \times 55$, within-cluster index fanout $= 4$, leaf size $= 4$; Desired number of nearest neighbors $= 20$, actual number of retrieved samples adjusted to maintain Recall between .95 and .96

index only the first few dimensions (from 4 to 8, depending on the number of clusters), that contribute to a large part of the variability of the data. Candidate nodes and leaves are therefore determined in 4- to 8-dimensional spaces, while distances between the query and the vectors stored at the leaves are computed in 10, 20 or 30 dimensions. Due to this difference, the number of candidate nodes visited during the search increases with the retained number of dimensions. We shall see immediately that most of the additional candidates are internal nodes rather than leaves.

When the leaf nodes are stored on disk, and cached only when required, the search cost is essentially determined by the I/O, and the speedup with respect to exhaustive scan of the entire database is essentially equal to the reciprocal of the fraction of the database contained in the visited leaves. This last quantity is shown, as a percentage, Figure 14(b). Note, first, that the number of retained dimensions has a secondary effect, indicating that the number of visited leaves does not change significantly with the number of retained dimensions. The number of clusters, however, has a major effect. For example, when retaining 20 dimensions on average, a CSVD index based on 2 clusters visits on average

7% of the database in response to a query, while an index based on 32 clusters visits on average .7% of the database, which corresponds to a 10-fold gain.

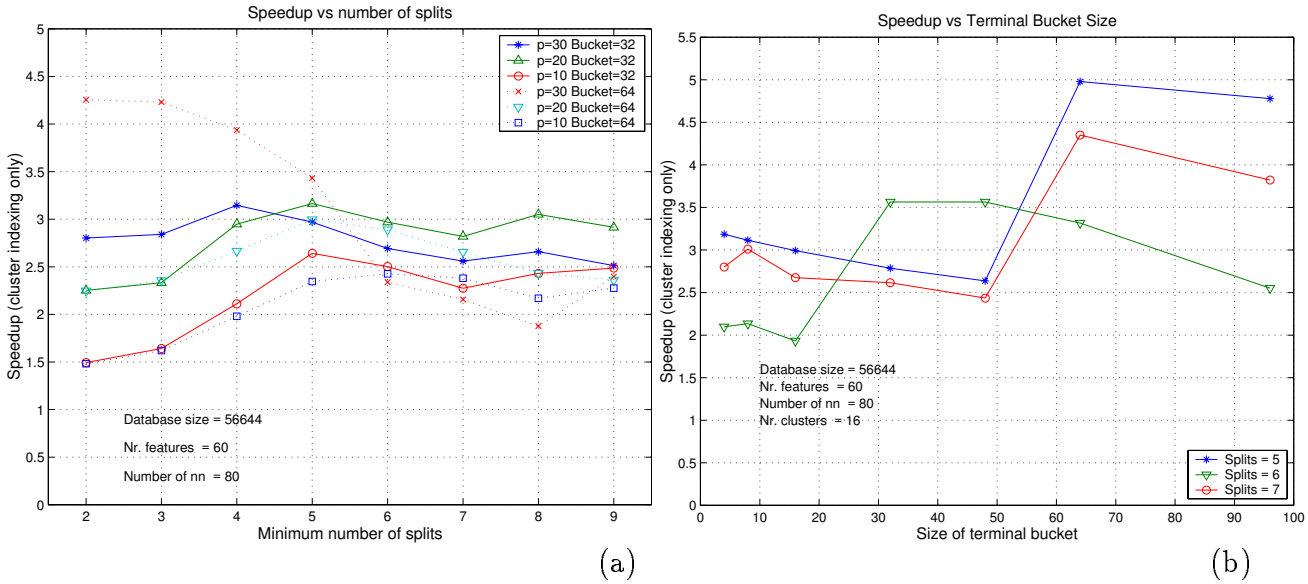

Figure 15: (a) Speedup due to within cluster index as a function of fanout, parameterized by the number of retained dimensions p and the bucket size; (b) Speedup due to within cluster index as a function of the terminal bucket size, parameterized by the fanout, for 16 clusters. Table size $=56,688 \times 60$, desired number of nearest neighbors $= 80$, actual number of retrieved samples adjusted to maintain Recall between .95 and .96

Additional design points are the size of the terminal leaves and the desired fan-out at internal nodes. We have found that leave sizes between 32 and 64 combined with fan-out between 4 and 6 produce the best results on our databases (Figure 15).

5 Summary

Many emerging multimedia database and data mining applications require efficient similarity retrieval techniques from very high dimensional datasets. In this paper, a novel methodology, *Clustering Singular Value Decomposition (CSVD)*, is proposed for approximate indexing of numeric tables with high dimensionality. This method achieves additional dimensionality reduction than those based on SVD for a fixed NMSE by exploiting the local structure of heterogeneous datasets. In the proposed method, the feature space is

first partitioned into multiple subspaces and a dimension reduction technique such as SVD is then applied to each subspace. The CSVD index is constructed by recording the centroid of each cluster, the cluster radius, and its projection matrix. Search is performed by alternating the identification of the best unsearched candidate cluster to a within-cluster search step, until no candidate remain. A cluster is a candidate if it can contain samples that are closer to the query template than some of the currently retrieved results. Candidate clusters are identified with a simple but fast method requiring only the computation of the squared distance between query template and cluster centroid (computed at query-execution time) and of the squared radius of the cluster (computed while constructing the index).

Within-cluster search is performed on reduced dimensionality versions of the query template and cluster data-points, using an existing indexing structure and correcting for the distance between the query sample and the cluster subspace. Since SVD sorts the dimensions of the rotated space in order of decreasing variance, the indexing can be limited to the first few dimensions whenever they capture a significant portion of the variability.

The effectiveness of the proposed CSVD method is validated with datasets consisting of feature vectors extracted from three benchmark databases containing feature vectors extracted from remotely sensed images. With similar precision vs. recall performance, it is shown in this paper that the CSVD method has a significantly better fidelity to the original data than SVD for a 20-fold dimensionality reduction. Conversely, for fixed values of NMSE, search using CSVD significantly faster than linear scan, especially as the database size grows. For moderate size databases containing 160,000 55-dimensional records, we have observed an 140-fold increase in speed while maintaining a recall of better than 95% when retrieving 20 nearest neighbors.

Several other dimensions of the index design have been explored in the paper, such as the selection of the number of actually retrieved results to ensure a desired recall value, and the dependence of the speed on the within-cluster index parameters.

Several extensions to this work are currently in progress, in particular we are exploring the performance of SVD-friendly clustering methods, i.e., methods that create clusters with an eye on reduced dimensionality. We are also investigating the effect of multi-level clustering and SVD on performance and develop procedures for constructing an optimal index in this case.

Acknowledgements

We would like to wholeheartedly thank Dr. Philip S. Yu and Dr. Charu C. Aggarwal for the insightful discussions, precious suggestions, useful references and for their encouragement. We would also like to thank an anonymous Japanese patent examiner for pointing us to the essential reference [35].
This work was funded in part by grant no. NASA/CAN NCC5-101.

References

[1] C.C. Aggarwal and P.S. Yu. Finding generalized projected clusters in high dimensional spaces. In *Proc. 2000 ACM SIGMOD Int'l Conf. on Management of Data*, page To appear, May 14–19 2000.

[2] M. Beatty and B.S. Manjunath. Dimensionality reduction using multi-dimensional scaling for content-based retrieval. In *Proc. IEEE Int'l Conf. Image Processing, ICIP '97*, volume 2, pages 835–838, Santa Barbara, CA, October 1997.

[3] N. Beckmann, H.P. Kriegel, R. Shneider, and B. Seeker. The R*−tree: an efficient and robust access method for points and rectangles. In *Proc. 1990 ACM SIGMOD Int'l Conf. on Management of Data*, pages 322–331, Atlantic City, NJ, USA, May 23-25 1990.

[4] S. Berchtold, C. Bohm, B. Braunmueller, D.A. Keim, and H.-P. Kriegel. Fast parallel similarity search in multimedia databases. In *Proc. 1997 ACM SIGMOD Int'l Conf. on Management of Data*, pages 1–12, Tucson, AZ, 12-15 May 1997.

[5] S. Berchtold, C. Bohm, D.A. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proc. 16th ACM ACM Symp. Principles of Database Systems, PODS '97*, pages 78–86, Tucson, AZ, May 1997.

[6] S. Berchtold, D.A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proc. 19th Int'l Conf. on Very Large Data Bases VLDB '93*, pages 28–39, Bombay, India, September 1996.

[7] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proc. Int'l Conf. Database Theory (ICDT '99)*, pages 217–235, Jerusalem, Israel, 1999.

[8] S. Blott and R. Weber. A simple vector-approximation file for similarity search in high-dimensional vector spaces. Technical report, Institute of Information Systems, ETH, Zurich, Switzerland, 1997.

[9] V.S. Cherkassky, J.H. Friedman, and H. Wechsler. *From Statistics To Neural Networks: Theory and Pattern Recognition Applications*. Springer-Verlag, 1993.

[10] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Soc. B*, 39(1):1–38, 1977.

[11] B.S. Everitt. *Cluster Analysis*. John Wiley & Sons, 3rd edition, 1993.

[12] C. Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Publishers, Boston, 1996.

[13] C. Faloutsos and K.-I. Lin. FastMap: a fast algorithm for indexing, data-mining, and visualization of traditional and multimedia datasets. In *Proc. 1995 ACM SIGMOD Int'l Conf. on Management of Data*, pages 163–174, San Jose, CA, May 1995.

[14] B.V. Gaede and O. Gunther. Mutidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, June 1998.

[15] A. Guttman. R-trees: a dynamic index structure for spatial searching. *ACM SIGMOD Record*, 14(2):47–57, June 1984.

[16] T.Y. Hou, P. Liu, A. Hsu, and M.Y. Chiu. Medical image retrieval by spatial feature. In *Proc IEEE Int'l Conf. Systems, Man, and Cybernetics*, pages 1364–1369, 1992.

[17] I.T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

[18] B.S. Kim and S.B. Park. A fast k nearest neighbor finding algorithm based on the ordered partition. *IEEE Trans. Pattern Analysis and Machine Intelligence*, PAMI-8(6):761–766, November 1986.

[19] F. Korn, H.V. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *Proc. 1997 ACM SIGMOD Int'l Conf. on Management of Data*, pages 289–300, Tucson, AZ, May 1997.

[20] R. Kurniawatio, J.S. Jin, and J.A. Shepherd. The SS+-tree: An improved index structure for similarity searches. In *Proc. SPIE - Int. Soc. Opt. Eng.*, volume 3022, *Storage Retrieval Image Video Databases V*, pages 110–120, February 1997.

[21] C.-S. Li and V. Castelli. Deriving texture feature set for content-based retrieval of satellite image database. In *Proc. of the IEEE Int'l Conf. Image Proc.*, pages 567–579, Santa Barbara, CA, Oct. 26-29 1997.

[22] C.-S. Li, V. Castelli, and L.D. Bergman. Progressive content-based retrieval from distributed image/video databases. In *Proc. 1997 IEEE Int'l Symp. Circuit Systems ISCAS97*, volume 2, pages 1484–87, Hong Kong, June 9-12 1997.

[23] Y. Linde, A. Buzo, and R.M. Gray. An algorithm for vector quantizer design. *IEEE Trans. Communications*, COM-28(1):84–95, January 1980.

[24] A. Pentland, R.W. Picard, and S. Sclaroff. Photobook: Tools for content-based manipulation of image databases. In *Proc. SPIE - Int. Soc. Opt. Eng.*, volume 2185, *Storage Retrieval Image Video Databases*, pages 34–47, February 1994.

[25] L. M. Po and C. K. Chan. Tree search structures for image vector quantization. In *Proc. of Int'l Symposium Sig. Proc. and Appl. ISSPA 90*, volume 2, pages 527–530, Australia, August 1990.

[26] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C*. Prentice-Hall, 1993.

[27] J.T. Robinson. The k-d-b-tree: A search structure for large multidimensional dynamic indexes. In *Proc. 1981 ACM SIGMOD Int'l Conf. on Management of Data*, pages 10–18, 1981.

[28] N. Russopoulos, S. Kelley, and F. Vincent. Narest neighbor queries. In *Proc. 1995 ACM SIGMOD Int'l Conf. on Management of Data*, pages 71–79, San Jose, CA, May 1995.

[29] R. Samadani, C. Han, and L.K. Katragadda. Content-based event selection from satellite images of the aurora. In *Proc. SPIE - Int. Soc. Opt. Eng.*, volume 1908, *Storage Retriev. Image Video Datab.*, pages 50–59, February 1993.

[30] J.R. Smith and S.-F. Chang. VisualSeek: A fully automated content-based image query system. In *Proc. ACM Multimedia '96*, pages 87–98, Boston, MA, USA, Nov. 18-22 1996.

[31] T.R. Smith. A digital library for geographically referenced materials. *IEEE Computer Magazine*, 29(5):54–60, May 1996.

[32] S. W. Smoliar and H. Zhang. Content based video indexing and retrieval. *IEEE Multimedia*, 1(2):62–72, Summer 1994.

[33] A. Thomasian, V. Castelli, and C.-S. Li. RCSVD: Recursive clustering with singular value decomposition for dimension reduction in content-based retrieval of large image/video databases. Research Report RC 20704, IBM, 01/28/1997.

[34] W. Niblack et al. The QBIC project: Querying images by content using color texture, and shape. In *Proc. SPIE - Int. Soc. Opt. Eng.*, volume 1908, *Storage Retrieval Image Video Databases*, pages 173–187, 1993.

[35] Y.T. Young and P.S. Liu. Overhead storage consideration in a multilinear method for data file compression. *IEEE Trans. Software Engineering*, SE-6(4):340–347, July 1980.

A A Property of PCA

It is known that PCA minimizes the *normalized root mean square error metric* (NRMSE), among all procedures consisting of a linear transformation followed by variable subset selection. Here, the NRMSE is defined as [19]

$$NRMSE = \frac{\sqrt{\sum_{i=1}^{M}\sum_{i=1}^{N}(x_{i,j} - x'_{i,j})^2}}{\sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N}(x_{i,j} - \boldsymbol{\mu}_j)^2}}. \tag{8}$$

In this equation $(x_{i,j} - x'_{i,j})^2$ is the squared error for the $(i,j)^{th}$ row of the matrix, $x'_{i,j}$ is the approximation to $x_{i,j}$ value given by Equation (11) below, and $\boldsymbol{\mu}$ is the centroid of the collection of points. Thus, the numerator is the overall squared error introduced by the approximation. The denominator is the mean distance squared from the points to the centroid of the corresponding cluster times the number of points M. The $NRMSE^2$ is therefore the fraction of the total variance lost to dimensionality reduction, and, using the notation of Section 2.2 can be expressed as $NRMSE^2 = 1 - F_{var}$. In the following, assume without loss of generality that $\boldsymbol{\mu}$ is the origin of the coordinate axes.

When performing dimensionality reduction following SVD, the p columns of the column-orthonormal matrix $\mathbf{U}$ corresponding to the highest eigenvalues are retained [19].

The matrix $\mathbf{U}$ is obtained directly from Eq. (4). However, if eigenvalues and eigenvectors are obtained by decomposing the covariance matrix as $\mathbf{C} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$, (Equation

(5)) then $\mathbf{U}$ can be computed from the original data as $\mathbf{U} = \mathbf{XVS}^{-1}$, by post-multiplying Equation (4) with $\mathbf{VS}^{-1}$.

For a given number of retained dimensions p both methods introduce the same error as shown below. The method given in this paper is more efficient from the viewpoint of computational cost for NN queries, window queries, etc. In the case of NN queries, once the input or query vector is transformed and the appropriate p dimensions are selected, we need only be concerned with these dimensions. This is less costly than applying the inverse transformation in Eq. (4) to all the points considered by NN queries and evaluating Eq. (1) with N dimensions. A shortcut is possible in both cases by considering the squares of Euclidean distances and ignoring the remaining steps of a summation where the partial sum exceeds the squared distance of the k^{th} nearest neighbor.

To show that preserving p columns of the $\mathbf{Y}$ matrix results in the same NRMSE and F_{var} as preserving the same dimsensions of the $\mathbf{U}$ matrix.

Retaining the first p dimensions in the transformed matrix $\mathbf{Y} = \mathbf{XV}$

$$y_{i,j} = \sum_{k=1}^{N} x_{i,k} v_{k,j}, \qquad 1 \leq j \leq p,\ 1 \leq i \leq M. \tag{9}$$

while the rest of the columns of the matrix are set to the mean and are effectively ignored. The squared error in representing vector $\mathbf{x_i}$ with $\mathbf{y_i}$ is then

$$e_i = \sum_{j=p+1}^{n} y_{i,j}^2 = \sum_{j=p+1}^{n} \sum_{k=1}^{n} (x_{i,k} v_{k,j})^2, \qquad 1 \leq i \leq M. \tag{10}$$

When p dimensions of the $\mathbf{U}$ matrix are retained, the elements of $\mathbf{X}$ are approximated by truncating the summation to p terms

$$x'_{i,j} = \sum_{k=1}^{p} s_k u_{i,k} v_{j,k}, \qquad 1 \leq j \leq N,\ 1 \leq i \leq M. \tag{11}$$

Then the squared error in representing the i^{th} point, given $y_{i,m} = s_m u_{i,m}$, is

$$e'_i = \sum_{j=1}^{N} \sum_{k=p+1}^{N} (s_k u_{i,k} v_{j,k})^2 = \sum_{j=1}^{N} \sum_{k=p+1}^{N} (y_{i,k} v_{j,k})^2$$

The following few steps show that $e_i = e'_i, 1 \leq i \leq M$

$$\sum_{j=1}^{N}(\sum_{k=p+1}^{N} y_{i,k}v_{j,k})(\sum_{k'=p+1}^{N} y_{i,k}vj,k) = \sum_{k=p+1}^{N}\sum_{k'=p+1}^{N} y_{i,k}y_{i,k'}\sum_{j=1}^{N} v_{j,k}v_{j,k'} =$$

$$\sum_{k=p+1}^{N}\sum_{k'=p+1}^{N} y_{i,k}y_{i,k'}\sum_{j=1}^{n}\delta_{k,k'} = \sum_{k=p+1}^{N}(y_{i,j})^2 = \sum_{k'=p+1}^{N}\sum_{k=1}^{N}(x_{i,k}v_{k,j})^2 = e_i.$$

Note that $\delta_{k,k'} = 1$ for $k = k'$ and $\delta_{k,k'} = 0$ for $k \neq k'$.