The VLDB Journal (2007) 16(2): 219–234
DOI 10.1007/s00778-005-0167-3

REGULAR PAPER

Heng Tao Shen · Xiaofang Zhou · Aoying Zhou

An adaptive and dynamic dimensionality reduction method for high-dimensional indexing

Received: 28 July 2004 / Accepted: 10 May 2005 / Published online: 20 October 2005

Abstract The notorious "dimensionality curse" is a well-known phenomenon for any multi-dimensional indexes attempting to scale up to high dimensions. One well-known approach to overcome degradation in performance with respect to increasing dimensions is to reduce the dimensionality of the original dataset before constructing the index. However, identifying the correlation among the dimensions and effectively reducing them are challenging tasks. In this paper, we present an adaptive *Multi-level Mahalanobis-based Dimensionality Reduction* (MMDR) technique for high-dimensional indexing. Our MMDR technique has four notable features compared to existing methods. First, it discovers elliptical clusters for more effective dimensionality reduction by using only the low-dimensional subspaces. Second, data points in the different axis systems are indexed using a single B^+-tree. Third, our technique is highly scalable in terms of data size and dimension. Finally, it is also dynamic and adaptive to insertions. An extensive performance study was conducted using both real and synthetic datasets, and the results show that our technique not only achieves higher precision, but also enables queries to be processed efficiently.

Keywords High-dimensional indexing · Dimensionality reduction · Correlated clustering · Subspace · Projection

1 Introduction

Many database applications, such as multimedia retrieval, exploratory data analysis, market basket application and timeseries matching, involve high-dimensional data. Indexing high-dimensional data has been an area of active research for a long time and many indexing techniques have been proposed [1]. However, the performance of these indexes degrades rapidly with increasing dimensionality [2].

One approach to minimize the effect of this "dimensionality curse" is to reduce the number of dimensions of the high-dimensional data before indexing it [3, 4]. The data is first transformed into a much lower-dimensional space using dimensionality reduction methods and then an index is built on it.

Transforming data from a high-dimensional space to a lower-dimensional space without losing critical information is not a trivial task. In this paper, we propose a dimensionality reduction technique called *Multi-level Mahalanobis-based Dimensionality Reduction* (MMDR) for indexing based on the following two observations. First, elliptical-shaped (correlated) clusters are more suitable for dimensionality reduction than spherical-shaped clusters. Second, we observe that certain levels of the lower-dimensional subspaces may contain sufficient information for correlated cluster discovery in high-dimensional space. In MMDR, *Principal Component Analysis* (PCA) [5] is employed to find the lower dimensions for dimension reduction. Most of the information in the original space can be condensed into a few dimensions along which the variances in the data distribution are the largest. We make use of *Mahalanobis distance* (MahaDist) in our approach instead of the standard L-norm distance functions.

Mahalanobis distance could be applied to find ellipsoidal correlated data, by taking local elongation into account. Instead of equally treating all values, MahaDist weights the differences by the range of variability in the dimension of the data points. It weights the variation along the axis of elongation less than that in the shorter axis of the ellipse. It can be shown that the surfaces on which MahaDist is a constant are ellipses.

Euclidean distance-based clustering algorithms are not meant to discover elliptical shape, since the clusters identified are circular in shape. Figure 1 illustrates two clusters, one obtained using Euclidean distance and the other obtained by Mahalanobis distance. Point A is a valid point and

H. T. Shen. (✉)X. Zhou
School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane QLD 4072, Australia
E-mail: {shenht, zxf}@itee.uq.edu.au

A. Zhou
Department of Computer Science, Fudan University, China
E-mail: ayzhou@fudan.edu.cn

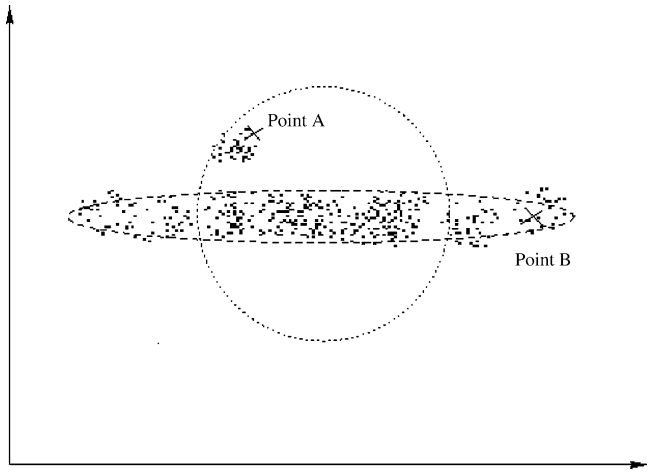


Fig. 1 Mahalanobis vs. Euclidean

Point B is a noise in the cluster of the circle if Euclidean distance is employed. However, in terms of Mahalanobis measurements, Point B has a substantially smaller distance to the centroid than Point A, since it lies along the direction of the group that has the largest variance. Thus, Point A is a noise, while Point B is valid. Therefore, while Euclidean distance-based algorithms produce circular subsets as shown in Fig. 1, Mahalanobis distance-based algorithms produce elliptical clusters where data points are well correlated and more natural for dimensionality reduction, as dimensions with large variances of data are kept and dimensions with small variances of data are eliminated.

Based on multi-level low-dimensional projections produced by PCA and the Mahalanobis distance function, MMDR can quickly identify highly correlated elliptical clusters. After dimensionality reduction, each cluster of data is in a different axis system. Instead of creating one index for each cluster, we build one index for all the clusters for K nearest neighbor (KNN) queries. We extend a recently proposed B^+-tree-based index – iDistance [1, 6], to index the data projections from the different reduced-dimensionality spaces. The extended iDistance allows us to index data points from different axis systems in a single index efficiently. Performance studies using real and synthetic datasets were conducted to evaluate the effectiveness and precision of the technique. The results show significant performance gain over an existing method [4]. Experiments on datasets with very high dimensionality (up to 200 dimensions) and datasets with dynamic insertions show that the proposed method is scalable in terms of both size and dimensionality, and is able to adapt to dynamic insertions.

A preliminary version of this paper appeared in [7]. There we stressed the algorithm of MMDR. Here in this paper, we extended the work in several ways. First, we present the more detailed algorithm of Scalable MMDR. Second, and more importantly, we introduce the algorithm of dynamic MMDR to handle dynamic updating. Third, more experimental results are reported by using much larger real-life datasets. Fourth, two more traditional dimensionality reduction methods and two more index accessing methods are also investigated and compared. Fifth, the effect of dynamic insertion and noise is also reported in the experiment.

The rest of the paper is organized as follows. In Sect. 2, we present some related works. In Sect. 3, we provide the definitions for using PCA and Mahalanobis distance in dimensionality reduction. We present our MMDR algorithm and its variant in Sect. 4. In Sect. 5, we propose the extended iDistance for indexing data points in reduced-dimensionality spaces and for handling dynamic insertions. Experiments are presented in Sect. 6 and conclusion is drawn in Sect. 7.

2 Related work

In dimensionality reduction for indexing, [4] proposed two strategies. In the first strategy, called *Global Dimensionality Reduction* (GDR), all the data as a whole is reduced down to a suitable dimension on which search time and access costs are optimized. This strategy is unable to handle datasets that are not globally correlated. The other strategy, called *Local Dimensionality Reduction* (LDR), divides the whole dataset into separate clusters on the basis of the correlation of the data, and then indexes each cluster separately. Unfortunately, LDR is not able to detect all correlated clusters effectively because it does not consider correlation or dependency between dimensions. After performing dimensionality reduction on each cluster, a much lower-dimensional subspace corresponding to each cluster is generated, and each subspace is in different axis system. To index the reduced subspaces, a Global indexing structure [4] was introduced, which consists of separate high-dimensional indices for every subspace. All these indices are connected to a single root node, which serves the purpose of a global directory.

High-dimensional structures have been the focus of extensive research [8]. Recent proposals can be categorized into the following two approaches: *data approximation* and *data transformation*. The technique of representing of the original data points using smaller and approximate representations is referred as data approximation. The VA-File [9] is one of typical proposals. The VA-File (Vector Approximation file) represents the original data points by much smaller vectors in form of bit sequences. The main drawback of the VA-File, however, is that it defaults in assessing the full distance between the approximate vectors, which imposes a significant overhead, especially if the underlying dimensionality is very large. Moreover, the VA-File does not adapt gracefully to highly skewed data. Data transformations provide another direction for high-dimensional indexing. Such techniques include the Pyramid technique [10], iDistance [6], etc. The Pyramid technique [10] divides the D-dimensional data space into two-dimensional pyramids and then cuts each pyramid into slices each of which forms a data page. It provides a mapping from D-dimensional space to single-dimensional space. iDistance [6] transforms a high-dimensional point into a single-dimensional distance

value with reference to its corresponding reference point. They suffer, however, from the fact that the search involves comparing distances between the full high-dimensional representation of the data points; thus, pruning during search becomes problematic as the dimensionality increases.

Clustering algorithms have been studied recently in the domain of data mining and pattern discrimination. Methods proposed for high-dimensional data clustering are related to our work. PROCLUS [11] clusters data according to the correlation among the data along certain original dimensions. OptGrid [12] finds clusters in a high-dimensional space by projecting data onto each axis and partitioning the data by using cutting planes at low-density points. Techniques based on the wavelet transform [13] and discrete cosine transform [14] rely on the partitioning of data space into grids similar to OptGrid. Wavelet transform [13] and discrete cosine transform [14] based techniques rely on the partitioning of the data space into grids similar to OptGrid. These approaches do not work well when well-separated clusters in actual space overlap after they are projected onto a certain axis.

Ref. [15] presents various results of qualitative behaviors of L-norm distance matrices for measuring proximity in high-dimensional spaces, and examines the meaningfulness of similarity in such spaces. The study shows that clustering quality and answer sets vary from one distance metric to another. In this paper, we examine a different distance function, the Mahalanobis function [16], to explore the local intrinsic cluster shape for elliptical clusters discovery (which cannot be detected by L-norm functions). Mahalanobis distance has been used in face detection to discover actual non-isotropic face patterns among thousands of face images using a k-means like algorithm called the elliptical k-means method [17]. It is a nested loop algorithm, where the inner loop performs k-means using Mahalanobis distance and the outer loop re-computes the covariance matrix of each cluster. Both loops stop when there is no change to the cluster membership.

3 Definitions

In this section, we provide the basic definitions.

Definition 1 (Ellipticity)
Ellipticity(e) is the deviation of an ellipse or an ellipsoid from the form of a circle or a sphere, which is the ratio of the difference of the two sub-axes to the minor axis:

$$e = \frac{b - a}{a}$$

where b is the radius along the major axis and a is the radius along the minor axis, as shown in Fig. 2.

Assume all the points are clustered inside the ellipse. To reduce the dimensionality, all the points are projected onto the major axes. Thus, the minor axes can be eliminated. Obviously, the larger the e is, the more effective dimensionality reduction can be obtained. When $e = 0$, the data points form a circle, and the dimensionality reduction technique becomes ineffective.

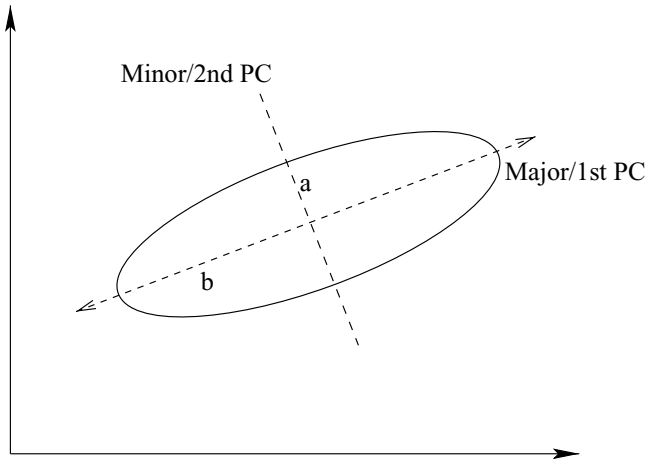


Fig. 2 Illustration of ellipticity

Definition 2 (Mahalanobis distance)
The *covariance* of data in two feature spaces measures their tendency to vary together. In a multi-dimensional space, the variance measures the relative 'radius' of a cluster along each dimension, and the covariance indicates the orientation of the cluster. Both the variance and the covariance co-determine the shape of the cluster. Collecting them together, we get the covariance matrix **C**. Now let us look at the distance function called *Mahalanobis distance* by using the inverse of covariance matrix.

Given a cluster centered at O, the *Mahalanobis distance* between a point P and O is given as follows:

$$\text{MahaDist}(P, O) = (P - O)^T C^{-1} (P - O)$$

where **C** is the covariance matrix describing the cluster's shape.

From the Mahalanobis distance, we get a normalized measure: *Normalized Mahalanobis distance*.

$$\text{MahaDist}_n(P, O) = \frac{1}{2}(d \ln(2\Pi \cdot \mid C \mid) + (P - O)^T C^{-1} (P - O))$$

where d is the dimensionality, Π is the trigonometric number 3.14 and $\mid C \mid$ is the determinant of **C**. Notice that given a spatial displacement between a point and an ellipsoid, the standard Mahalanobis distance tends to be smaller for long clusters with large covariance matrices than for small clusters. With standard Mahalanobis distance, the larger cluster will keep increasing in size, and eventually overwhelm the smaller clusters. Normalized Mahalanobis distance avoids such situations [17].

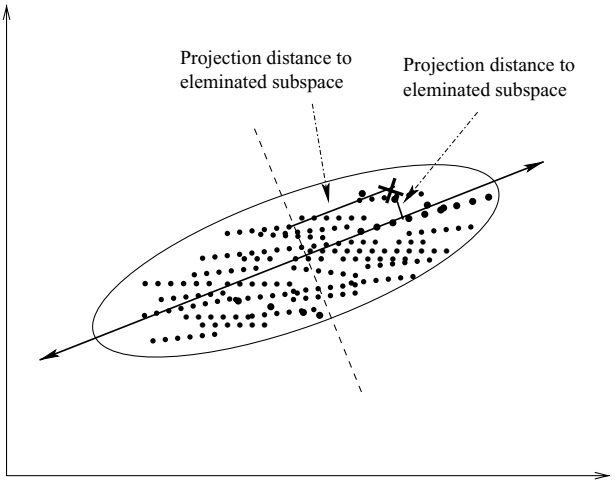


Fig. 3 Two projection distances

Definition 3 (Multi-level projections)

PCA [5] examines the variance structure in the dataset and determines the directions along which the data exhibits high variance. The first principal component is the eigenvector corresponding to the largest eigenvalue of the dataset's covariance matrix **C**, the second component corresponds to the eigenvector with the second largest eigenvalue and so on. It is interesting that the principal components in PCA are just the eigenvectors of the covariance matrix in the Mahalanobis distance which describes the dataset's shape. An example is shown in Fig. 3, where the preserved dimension is the first principal component, and the eliminated dimension is the second principal component. In dimensionality reduction, given a point P in a dataset, there are two projections. One is the projection on the preserved subspace P' that we are interested in; the other is the projection on the eliminated subspace P''. The d_r-dimensional projection P'_{d_r} can be defined as:

$$P'_{d_r} = P \cdot \Phi_{d_r}$$

where Φ_{d_r} represents the matrix containing first to d_rth principal components. Changing d_r with different values, we can generate multi-level projections of the data for cluster discovery purposes in the *MMDR* algorithm.

Definition 4 (Projection distance)

Continuing with the previous two projections, $ProjDist_r$ measures the distance from P to P' and $ProjDist_e$ measures the distance from P to P'' on the eliminated subspace. More specifically, $ProjDist_r$ is the information lost from the original representation P to its reduced d_r-dimensional representation P'. $ProjDist_e$ is the information retained. Figure 3 illustrates the two projection distances. In the following paragraphs, *ProjDist* represents $ProjDist_r$.

Based on the previous two projection distances, we extend the definition of *ellipticity* to multi-dimensional space as:

$$e = \frac{Max(ProjDist_e) - Max(ProjDist_r)}{Max(ProjDist_r)}$$

Table 1 Table of symbols and default values

Symbols	*Descriptions*	*Value*
N	Data size	
d	Original dimensionality	
d_r	Optimal dimensionality	
s_dim	Subspace dimensionality	
e	Ellipsoid's ellipticity	
r	Mahalanobis radius	
C	Covariance matrix	
$ProjDist_r$	Distance to remaining subspace	
$ProjDist_e$	Distance to eliminated subspace	
MPE	Mean Projection Error	
σ	Outlier set	
β	$ProjDist_r$ Threshold	0.1
$MaxMPE$	Maximum MPE Allowed	0.05
EC	Elliptical cluster	
$MaxEC$	Maximum EC allowed	10
$MaxDim$	Maximum remaining dimensions allowed	20
ε	Data stream percentage	0.005
ξ	Outlier percentage	0.005
k	Num of IDs in lookup table	3

where $Max(ProjDist_e)$ is the radius along the remaining subspace, and $Max(ProjDist_r)$ is the radius along the eliminated subspace. The cluster's Mahalanobis radius r is $Max(ProjDist_r)$. For dimensionality reduction, the larger the *ellipticity* value, the more effective dimensionality reduction can be performed.

Definition 5 (Mean ProjDist$_r$ Error (MPE))

Mean $ProjDist_r$ Error is defined as the average representation error when points are mapped from the original space to the eliminated subspace.

$$\text{MPE} = \frac{\sum_{i=1}^{N} ProjDist_r(P_i, O)}{N}$$

This parameter indicates the average information lost after performing dimensionality reduction. It is used in our algorithm to control the user-defined maximal error allowed.

Table 1 gives a summary of the symbols and their respective description with default values used in the experiments.

4 Multi-level Mahalanobis-based Dimensionality Reduction

In this section, we present our MMDR algorithm, followed by its cost analysis and optimization. Finally, one variation of MMDR called Scalable MMDR is also introduced to handle very large dataset.

4.1 MMDR algorithm

The MMDR algorithm, which is outlined in Fig. 4, consists of two major steps, namely: *Generate Ellipsoid* and *Dimensionality Optimization*.

MMDR Algorithm

Generate Ellipsoid (GE)

```
Variable: ellipsoid_array, subspaces;
GE(data, d, s_dim)
1.    projections ← getProj(data, s_dim);
2.    semi_ellip ← ellip_k_means(projections,s_dim);
3.    // process each semi_ellips
4.    for each semi_ellip with size > 0
5.          semi_ellip_data ← restoreData(semi_ellip);
6.          semi_ellip ← getProj(semi_ellip_data, s_dim);
7.          MPE ← getMPE(s_dim);
8.          if MPE > MaxMPE and 2*s_dim > d
9.              GE(data, d, 2*s_dim);
10.         else
11.             add semi_ellip_data into ellipsoid_array
```

Dimensionality Optimization

```
12.   for each ellipsoid_array[i]
13.         d_r ← min(MaxDim, s_dim);
14.         MPE ← getMPE(d_r);
15.         while change of MPE < threshold
16.             d_r −− ;
17.             MPE ← getMPE(d_r);
18.         projections ← getProj(ellipsoid_array[i],d_r);
19.         for each projection
20.             ProjDist ← getProjDist();
21.             if ProjDist ≤ β
22.                 add it to this subspace;
23.             else
24.                 add it to noise set
```

Fig. 4 MMDR algorithm

In *Generate Ellipsoid*, we recursively apply multi-level projections from low- to high dimensionality until the ellipsoids are all discovered. At each level, Mahalanobis distance is applied to detect possible ellipsoids. Any unqualified ellipsoid is passed to *Generate Ellipsoid* with a higher subspace dimensionality so that more information can be used for clustering. We adopt this divide-lower-before-conquer-upper approach on the basis of the following observations. First, in high-dimensional space, some dimensions may contain little information, which may not be very helpful when it comes to identifying cluster membership. Second, for well-separated clusters in the subspace, their correspondences in the higher-dimensional space are usually well separated because of the property of PCA. In our algorithm, MPE indicates how much information is lost during the projection process. It is used as the parameter to determine if the subspace projections carry enough information to reflect the shape of their correspondence in the original space.

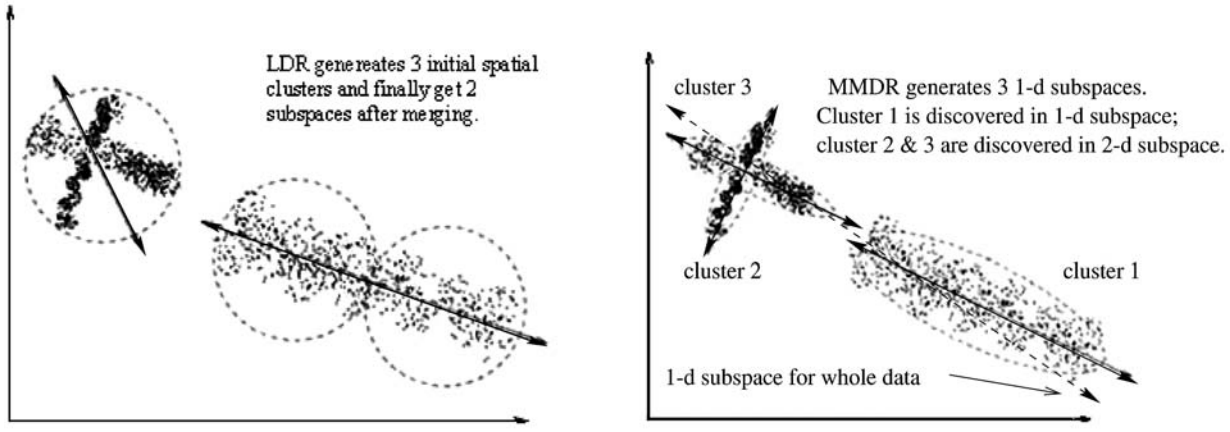
Generate Ellipsoid is invoked with a small subspace dimensionality – *s_dim*. In line 1 of Fig. 4 the low-dimensional projections are produced from the original d-dimensional space, followed by elliptical k-means clustering in this low-dimensional subspace, line 2. The data are then partitioned into semi-ellipsoids at *s_dim*-dimensional subspace. We call this *semi-ellipsoid*, since we have not decided yet whether it properly indicates the shape of its correspondence in the original space. From lines 3 to 11, each semi-ellipsoid is handled individually. For each semi-ellipsoid discovered earlier, its corresponding shape is restored in the original dimensional space (line 5), and its local *s_dim*-dimensional subspace is generated (line 6). The newly produced projections are local to individual semi-ellipsoid and different from the projections produced in line 1. At line 7, the MPE to *s_dim*-dimensional subspace is computed.

If a semi-ellipsoid has an MPE smaller than the maximum error allowed, it suggests that the *s_dim*-dimensional subspace can approximately represent its original data. Otherwise, there are two possible reasons for the big MPE. First, it could be due to the overlap of several clusters in the subspace such that each point does not project to its local subspace. Higher subspace dimensionality should be retained in order to distinguish each cluster. Second, though it is a single cluster, the *s_dim* could be too small for a subspace to represent original dimensional data. To further discover ellipsoids in each semi-ellipsoid, we increase the *s_dim* twofold without losing generality and recursively call *Generate Ellipsoid* (line 9). Therefore, the semi-ellipsoid is repeatedly partitioned locally. This step produces possible ellipsoids. It should be noted that the process of discovering ellipsoids in subspaces is the first step of dimensionality reduction, where the remaining subspace dimensionality of each ellipsoid at this stage is their respective s_dim, and further dimensionality optimization is performed in the next step.

Since the earlier step produces possible ellipsoids in their respective *s_dim*-dimensional subspaces, and ellipsoids are effective for dimensionality reduction, the *s_dim* of the subspaces discovered in *Generate Ellipsoid* can be further reduced (it should be noted that each ellipsoid may correspond to a different *s_dim* value). That is, if the change of MPE is less than the pre-set threshold, we decrease the dimensionality by 1, and the process is repeated till the aforementioned condition is false (lines 15–17). The final dimensionality is treated as the 'optimal' one and denoted as d_r. The points are projected into this d_r-dimensional subspace (line 18). A threshold value β is employed to determine whether a point belongs to a cluster. If the projection distance on $(d - d_r)$-dimensional eliminated subspace for a point is greater than β, this point is taken as an outlier (lines 23–24). Otherwise, it is classified as a member of the subspace (lines 21–22).

The final output of the algorithm is a set of subspaces and outliers. Each subspace may have a different optimal number of reduced dimensions. The outlier set remains in the original space, since its data points are not well correlated.

In a dataset, some clusters are elongated along certain directions and yet they are locally correlated. Such elongation may be detected in its lower-dimensional subspaces. Consider a two-dimensional subspace as shown in Fig. 5 projected from a higher-dimensional space (say four dimensional). This two-dimensional subspace can represent the original dimensional space with very little information being lost. The LDR technique [4] is able to discover correlated clusters on the original four-dimensional space and produce 2 one-dimensional subspaces as shown in Fig. 5a.



(a) Clusters Generated by LDR Using Euclidean Distance

(b) Clusters Generated by MMDR Using Mahalanobis Distance

Fig. 5 LDR vs. MMDR

In order to partition the bigger shape cluster for dimensionality reduction, the clustering radius must be suitably large. However, this will result in smaller clusters being grouped together as one. Obviously, substantial information is lost with the smaller shaped clusters. The situation is even worse for small-shaped clusters with a high density.

Figure 5b shows 3 one-dimensional subspaces produced by the MMDR algorithm. MMDR first projects the original dimensional space into one-dimensional subspace, then the elliptical k-means method partitions one-dimensional projections of whole data into two partitions: cluster 1's one-dimensional subspace and cluster 2 and 3's one-dimensional subspace. After restoring cluster 1's one-dimensional subspace to the original dimensional space and performing local one-dimensional projections (lines 5–6), MMDR detects that it is an ellipsoid, since its MPE is small. The one-dimensional subspace projected from clusters 2 and 3 overlaps heavily with the high MPE and thus its corresponding full dimensional shape/data is passed to *Generate Ellipsoid* by increasing the subspace dimensionality to 2. In the two-dimensional subspace, these two ellipsoids can be discovered by the Mahalanobis function. Dimensionality reduction is further performed in *Dimensionality Optimization* so that both can be reduced to one-dimensional subspaces with less information lost than if the LDR method were used.

In summary, MMDR has the following advantages. First, the ellipsoids can be effectively discovered at the subspace level of the data, rather than in the original space. Second, the ellipsoids can be discovered as soon as the shapes can be identified. Third, using Mahalanobis distance, the cost to perform clustering can be reduced dramatically, since it is done in low-dimensional subspace.

4.2 Cost analysis on MMDR

The cost of MMDR comes mainly from the elliptical k-means method (line 2), which takes $O(Iter_{\text{out}} \times Iter_{\text{inn}} \times d\_sim^2 \times N \times MaxEC)$, where $Iter_{\text{out}}$ and $Iter_{\text{inn}}$ is the number of iterations for the outer and inner loop, respectively, and d_sim^2 comes from distance computation. However, in MMDR, the input dimensionality s_dim is very small compared to the original d. Meanwhile, the input data size N becomes smaller as s_dim increases, which leads to $Iter_{\text{out}}$ and $Iter_{\text{inn}}$ being reduced also. To further reduce the cost of MMDR, we use the cost reduction techniques described in the next section.

4.3 Cost optimization on MMDR

We note that the most time-consuming step of the MMDR algorithm is the Mahalanobis distance computation between centroids and data points in the elliptical k-means method. In this section, we reduce the computational cost by using the following techniques. The factor *MaxEC* can be reduced to a small number by avoiding the computing of all the distances between *MaxEC* centroids and a data point. We only need to re-compute the distance between the k most closest centroids which may change the membership of a point, where $k \ll MaxEC$. This is based on the following observations. First, if a data point is to be re-assigned to another cluster, that cluster is most probably the one with the closest distance except the current assignment. Second, in each iteration, only a small portion of data points may change their membership. As the converging process continues, the number of data points changing memberships decreases quickly. Third, some data points may never change their membership.

A lookup table is designed to store the k most closest centroids' IDs computed in the previous iteration for each data point. In the next iteration, only those centroids whose IDs are stored in the lookup table are taken to compute and find the closest centroid. A data point entry in the lookup table is updated only when its membership is changed. By doing so, the factor *MaxEC* is removed from the overall cost.

Scalable MMDR Algorithm

1. Initialize the Ellipsoid Array;
2. Divide dataset into $\frac{1}{\varepsilon}$ data streams;
3. For each data stream
4. Pass it to *Generate Ellipsoids*;
5. Add newly generated ellipsoids into Ellipsoid Array;
6. Form a new data stream using all centroids in Ellipsoid Array;
7. Pass this new data stream to *Generate Ellipsoids*;
8. Re-assign point membership to the generated ellipsoids;
9. For each ellipsoid
10. Pass it to *Dimensionality Optimization*;

Fig. 6 Scalable MMDR algorithm

To further reduce the cost for large datasets, we introduce one additional field called *Activity* to the lookup table to indicate how frequently a data point changes its membership. It records the number of iterations that a data point does not change its membership. If the value of *Activity* is larger than a threshold, we say this data point is *inactive*, otherwise it is *active*. *Inactive* data points need not make any further distance computation and re-assignment unless the number of clusters is changed. This reduces the value of N dramatically at each iteration. Assume that at each iteration, only $\frac{1}{Iter_{\text{inn}}}$ of the dataset change their memberships, the factor of N is replaced by $\frac{N}{Iter_{\text{inn}}}$. As the converging process continues, the number of points which change their membership decreases dramatically. Therefore, the time complexity becomes $O(Iter_{\text{out}} \times d\_sim^2 \times N)$. Compared to LDR's time complexity of $O(N \times d^2 \times MaxEC)$, MMDR has a smaller dimensionality of d_sim than d, but with a larger factor of $Iter_{\text{out}}$ than $MaxEC$.

4.4 Scalable MMDR

For a very large dataset that cannot be completely loaded into the main memory buffer, the data scan at each iteration is extremely expensive. To make MMDR scalable for very large datasets, we introduce Scalable MMDR.

In Scalable MMDR, we divide the dataset into a number of data streams, which is defined as a sequence of data points read in order of indices, and we process one data stream at a time. A temporary array called Ellipsoid Array (EA) is created to store the ellipsoids' centroids generated for each data stream. Scalable MMDR loads a single data stream at a time and performs *Generate Ellipsoid* operation to generate small-size ellipsoids. These small ellipsoids' centroids are stored in the Ellipsoid Array. After all the data streams have been processed, only the Ellipsoid Array is in the buffer. By calling *Generate Ellipsoid* on the Ellipsoid Array, Scalable MMDR forms bigger sized ellipsoids by merging smaller ellipsoids whose centroids are stored in the Ellipsoid Array. Let the size of a data stream be ε percent of the data size, where ε is a very small positive value. The detailed algorithm is outlined in Fig. 6.

The key idea in Scalable MMDR is to summarize each small ellipsoid existing in data streams using its centroid. All centroids are then processed as a data stream whose size is very small. Since the size of the data stream is much smaller than the original data size N, empirically, it is reasonable to expect that $Iter_{\text{data stream}} \ll Iter_{\text{original data}}$. Hence, the total time required to cluster $\frac{1}{\varepsilon}$ data streams of size $\varepsilon \times N$ is generally much less then the time required to cluster N data points.

5 Indexing reduced subspaces

After dimensionality reduction, the projections in the reduced dimensionality subspaces have to be indexed using efficient indexes. Instead of using an index for each subspace, we want all the projections to be indexed in a single structure for ease of maintenance. We selected iDistance [1] as our base index due to its efficiency and its B^+-tree base structure.

The design of iDistance was motivated by two factors. One, the triangular inequality relationship enables the (dis)similarity between a query point and a data point to be derived with reference to a chosen reference point. Two, data points can be ordered on the basis of their distances to a reference point, and indexed on the basis of such distance value. This enables one to represent high-dimensional data in a single-dimensional space and use an existing B^+-tree. However, iDistance has to be extended to index subspaces in different axis systems and handle the dynamic insertion of data points.

5.1 Extended iDistance

The data partitioning strategy and reference point selection are straightforward, as the data partitions are determined by the MMDR algorithm and the centroid of each cluster is the ideal choice as the reference point. For each subspace (outliers as a subspace in its original dimensionality), all data points in subspaces are represented in a single-dimensional space with reference to its centroid of cluster. This is achieved by the following mapping function:

$$y = i \times c + dist(P, O_i)$$

where P is a data point in the subspace of ith ellipsoid EC_i, and O_i is its centroid. $dist(P, O_i)$ is the distance function that returns the distance between O_i and P. y is the index key for P. c is some constant to stretch the data range so that distance values are range partitioned on the basis of the reference points. That is, it serves to partition the single dimension space into regions so that points in the ith cluster will be mapped to the range $[i \times c, (i + 1) \times c]$.

Extended iDistance employs three data structures:

- A B^+-tree is used to index the transformed single-value points to facilitate speedy retrieval.
- An array is required to store the centroids and principal components of ellipsoids, and their respective nearest and farthest radii that define the subspace. This array is used for search purposes.

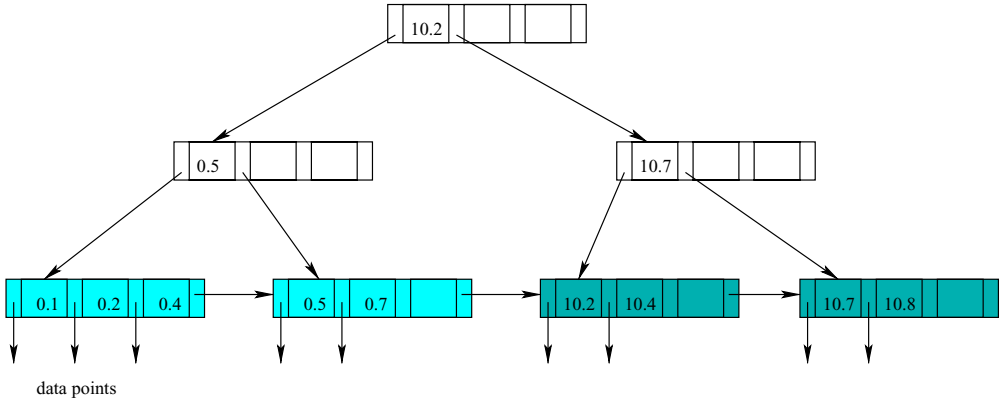


Fig. 7 A sample B^+-tree indexing two clusters

– An array is required to store covariance matrices of ellipsoids, the Mahalanobis radius, and the dimensionality retained. This array is used for the purpose of dynamic insertion.

Figure 7 depicts an example of two clusters indexed by a single B^+-tree, where the c is set as 10, the light shaded pages contain the keys of points from the first cluster and the dark shaded pages contain the keys of points from the second cluster.

To search for the K nearest neighbors of a query point q, the distance of the Kth nearest neighbor to q defines the minimum radius required for retrieving the complete answer set. Such a distance cannot be predetermined, and hence, an iterative approach that examines an increasingly larger sphere in each iteration has to be employed.

The algorithm works as follows. Given a query point q, finding K nearest neighbors (NN) begins with a query sphere defined by a *relatively small* radius R around q. For each cluster EC_i, the query point is mapped into q_i, which is the projection of q on the ith subspace.

Figure 8 shows an example with the maximum radius ranges of three clusters in the different axis systems, where EC_1 is in XY-plane, EC_2 in XZ-plane and EC_3 in YZ-plane. Here, for a query point q, its projection on three subspaces are q_1, q_2, and q_3 respectively. The shaded regions are the areas that need to be checked.

Searching in extended iDistance begins with the scanning of the auxiliary structure to identify the centroids whose data space (sphere area of cluster) overlaps with the query sphere defined by q_i and R. The search starts with a small global radius R for all subspaces, and step by step, the radius is increased to form a bigger query sphere. For each enlargement, there are three main cases to consider:

– The data space EC_i contains q_i. In this case, we want to traverse the data space sufficiently to determine the K nearest neighbors. This is done by first locating the leaf node where q_i may be stored. Since this node does not necessarily contain points whose distance is closest to q_i compared to its sibling nodes, we need to search left and right (inward and outward of data space) from the reference point accordingly. This situation is illustrated by the subspace EC_1 and q_1.
– The data space intersects the query sphere. In this case, we only need to search leftward (inward), since the query point is outside the data space. This situation is illustrated by the subspace EC_3 and q_3.
– The data space does not intersect the query sphere. Here, we do not need to examine the data space. This situation is illustrated by the subspace EC_2 and q_2.

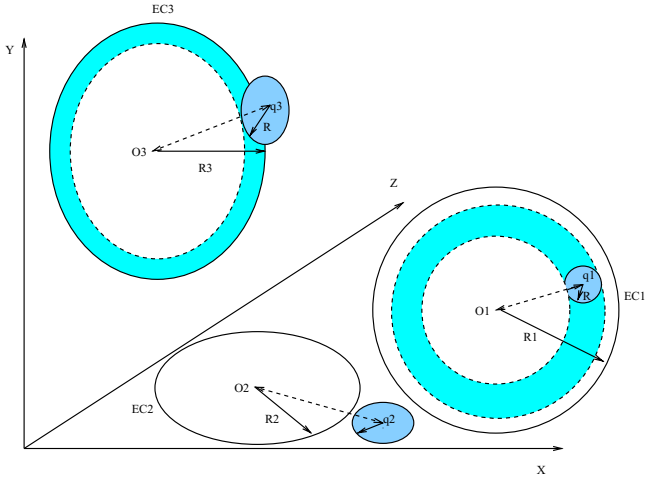


Fig. 8 Searching for NN Queries q_1, q_2 and q_3.

The search stops when the distance of the Kth NN object to q is less than search radius R. The search is correct as the distance between image query point and data point always lower bounds the actual distance between the actual query point and data point in the original space. The subspace search can be fast pruned by using the triangle inequality property:

$$\|Q - P\| \geq \|Q_j - P_j\| \geq \|Q_j - O_j\| - \|P_j - O_j\| \geq \|Q_j - O_j\| - R_j$$

where Q is query, P is original data point, Q_j is the projection in jth subspace, P_j the projection of P in jth subspace,

Dynamic MMDR Algorithm

```
1.  for each point P in data stream
2.          O ← nearestEllipsoid(P);
3.          dist_min ← minMahaDist(O);
4.          if dist_min ≤ O.r
5.              projection ← getProj(O.d_r);
6.              y ← getKey(O);
7.              insertIntoIndex(y);
8.              removeFromDataStream(P);
9.  subspaces ← MMDR(data stream);
10. for each subspace
11.         if merge(i)
12.             deleteFromIndex(i*c, (i+1)*c);
13.             form a new subspace with same ID - i;
14.         else
15.             Treat it as new subspace with new ID;
16. for each new subspace O_new
17.         for each point
18.             y ← getKey(O_new);
19.             insertIntoIndex(y);
```

Fig. 9 Dynamic MMDR algorithm

O_j the reference point in jth subspace, and R_j the maximum radius in jth subspace. $\|Q_j - O_j\| - R_j$ specifies the tightest searching bound for jth subspace.

5.2 Dynamic MMDR

In the real world, the database is not static, and it is too costly for a dimensionality reduction algorithm to re-analyze the whole database after some insertions. It is granted that for a very large database, a single data point insertion should not greatly affect the effectiveness of clustering and dimensionality reduction. However, over time, query accuracy and efficiency will be affected if the indexing mechanism is not adaptive to new data insertions.

To handle dynamic insertions, the MMDR algorithm is made to be adaptive to new insertions by splitting and merging subspaces if necessary. In the following algorithm, we treat a set of insertions as a batch insertion of a small dataset, which is the data stream described in Sect. 4.4.

For each data point P in the data stream, we first get the ellipsoid with minimal MahaDist to P and denote it as O (lines 2–3). If this distance is less than O's Mahalanobis radius – r, we get its projection in O's subspace (line 5), followed by mapping it to indexing value (line 6) and inserting it into the index (line 7), then removing it from the data stream. Next, the updated data stream is passed to MMDR to generate subspaces (line 9). For each subspace, if it has the same elongation and intersects with an existing subspace i, we delete all the entries whose indexing values are in the range of $[i * c, (i + 1) * c]$ (line 12) and form a new subspace with the same ID – i (line 13). Otherwise, we treat this subspace as a new subspace with a new ID (line 15). Finally, for each new subspace (lines 16–19), we map each data point into an indexing value with corresponding subspace, then insert it into the index.

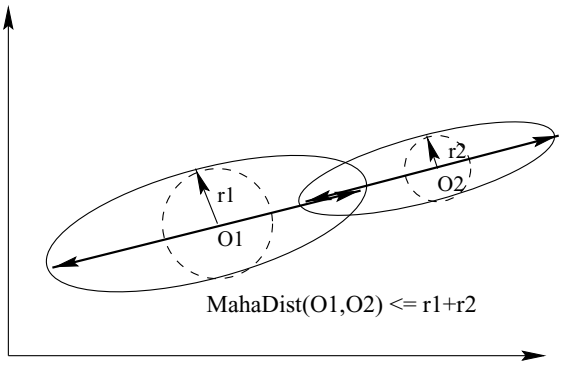


Fig. 10 Two ellipsoids intersect with same elongation

One important step in the algorithm shown earlier is to merge two subspaces when necessary. The following condition must be satisfied in order to merge two subspaces: their ellipsoids must have the same elongation and intersect each other. Figure 10 illustrates such a scenario. In Fig. 10, we only indicate the circumscribing ellipse of the cluster, and r is the Mahalanobis radius. If the MahaDist of two centroids of ellipsoids is not greater than the sum of their Mahalanobis radii, these two ellipsoids intersect. In this case, the two ellipsoids are merged into a new ellipsoid. Based on the new centroid, we re-map the points into indexing values with the existing ellipsoid's ID, followed by the standard insertion operation.

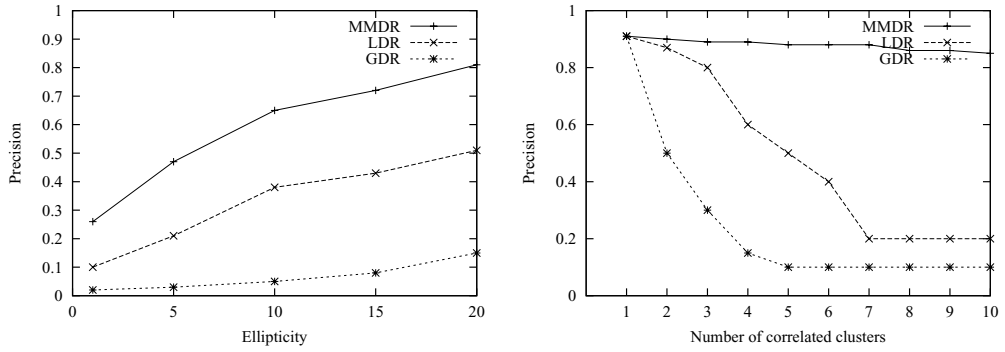
It is important to note that the algorithm does not require us to rebuild the whole tree. Instead, it only affects the partitions where merging is performed.

6 Performance study

In this section, we present the performance study that evaluates the effectiveness of MMDR and the efficiency of extended iDistance. For the experiments, we used the default values as shown in Table 1, and all experiments were done with an Ultra-10 SunOS 5.7 processor (333 MHz CPU and 256 MB RAM).

We have two categories of test data.

1. Real-life datasets: We have two small real-life datasets and one large real-life dataset. The first small dataset consists of 64-dimensional color histograms extracted from 70,000 color images from the Corel Database used in LDR [4]. The second small dataset consists of 64-dimensional Daubechies' wavelet [18] features extracted from 73,715 WWW images randomly crawled from over 40,000 websites. Wavelet features describe an image's shape, texture, and location information in a single representation. Here, we truncated the 64 most dominating wavelet coefficients as an image's visual feature. The large dataset consists of about 2.2 million frames



(a) Effect of Ellipticity (b)Effect of Number of Clusters

Fig. 11 Effect on precision

extracted from 3,000 video clips, which are TV advertisements captured by us from TV stations. Video clips (about 30 s average time length) are recorded by using Virtual Dub at PAL frame rate of 25 frames per second [19]. Each frame is represented by a 64-dimensional feature vector in the RGB color space.

2. Synthetic datasets: We have four sets of synthetic datasets: one small synthetic dataset containing 100,000 points in a 64-dimensional space, and three large synthetic datasets with 1,000,000 points each in a 50-, 100-, and 200-dimensional space. For each synthetic dataset, we used the algorithm (Appendix A) to generate correlated clusters in different subspaces with different distensibilities. Each subspace has its own size, orientation, and ellipticity.

We used 100 queries to obtain the mean precision and query cost on 10 NN, and L_2 distance was used for searching (note that Mahalanobis distance is used for discovering intrinsic ellipsoids, not for searching). The query precision is defined as follows:

$$Precision = \frac{R_{d_r} \bigcap R_d}{R_d}$$

where R_d and R_{d_r} are the results respectively returned from the original space and reduced subspaces.

6.1 Query precision

Here, dimensionality reduction methods serve the purpose of efficient indexing. However, they are lossy in nature. When a dimensionality method tries to reduce more, it may cause bigger loss of information and hence reduce query precision. Query precision is also affected by the correlation between data points and the number of correlated clusters. Here, we used the small dataset with 100,000 points.

Figure 11a shows the query precision with respect to increasing ellipticity. As we can see, the MMDR method performed much better than the LDR and GDR methods. The GDR method could achieve at most 15% of precision as the dataset is not globally correlated. As ellipticity decreased, LDR dropped faster than MMDR. Obviously, less correlation had more negative effect on the query precision of LDR than that of MMDR. In the next experiment, we varied the number of correlated clusters to test its effect on query precision. The results in Fig. 11b show that MMDR, LDR, and GDR all performed equally well when there was only one correlated cluster. But as the number of correlated clusters increased, MMDR was able to locate all correlated clusters effectively while maintaining its query precision. However, the query precision of LDR dropped rapidly, and so did that of the GDR method. This indicates that when clusters intersect and have different ellipticities and scales, LDR cannot discover all of them. As more such clusters exist, LDR performs worse. In contrast, the MMDR can discover the intrinsic number of correlated cluster based on Mahalanobis distance and thus is independent of the number of correlated clusters.

To see the effect of the number of eliminated dimensions on the effect of query precision, we conducted experiments using the small synthetic dataset and three real datasets. In this experiment, we set the maximum remaining subspace dimensionality $MaxDim$ as 20. Figures 12–14 present the effect of the number of dimensions retained after dimensionality reduction on the query precision on four datasets. In this experiment, we also compared traditional dimensionality reduction methods including Discrete Wavelet Transformation (DWT) and Discrete Fourier Transformation (DFT). The *conjugate property* of DFT was considered [20]. Notice that unlike the DWT coefficients that are real numbers, the DFT coefficients are complex numbers thus require twice the storage space. In line with [20], for the rest of this section, we assume that a DFT coefficient takes two floating numbers, while a DWT coefficient takes only one floating number. Given the same storage space and indexing structure, the number of coefficients of DWT that we can index is twice that of DFT's. For the sake of fair comparison with the same storage space and indexing structure, the dimensionality of DFT and DWT refers to the *number of float numbers* being indexed.

All the five methods showed increasing precision as the remaining dimensionality increased for the four datasets.

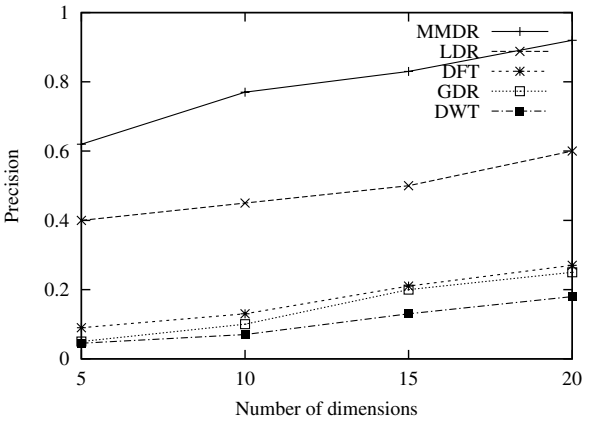


Fig. 12 Effect of dimensionality on query precision for small synthetic data

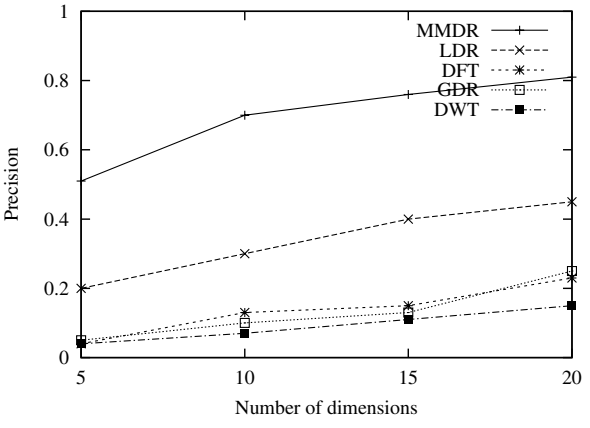


Fig. 13 Effect of dimensionality on query precision for color histogram

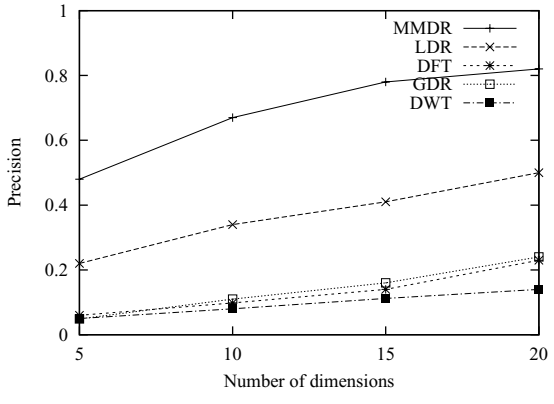


Fig. 14 Effect of dimensionality on query precision for video frames

However, MMDR achieved much higher precision. As shown in Fig. 12 for the synthetic dataset, at 20 dimensions, LDR could only achieve at most 60% of precision, and GDR could not achieve more than 25% of precision due to uncorrelated property. DFT performed similar to GDR, while DWT performed even worse than GDR since both are not meant to identify the correlations among dimensions.

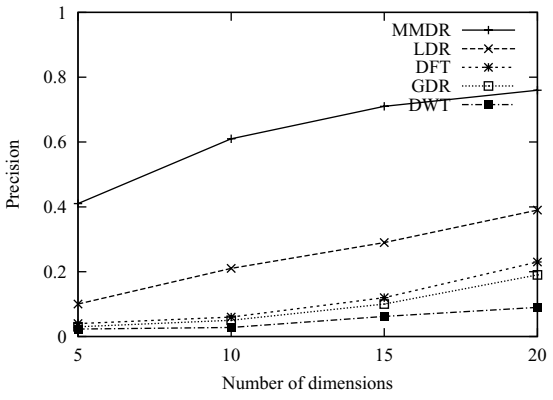


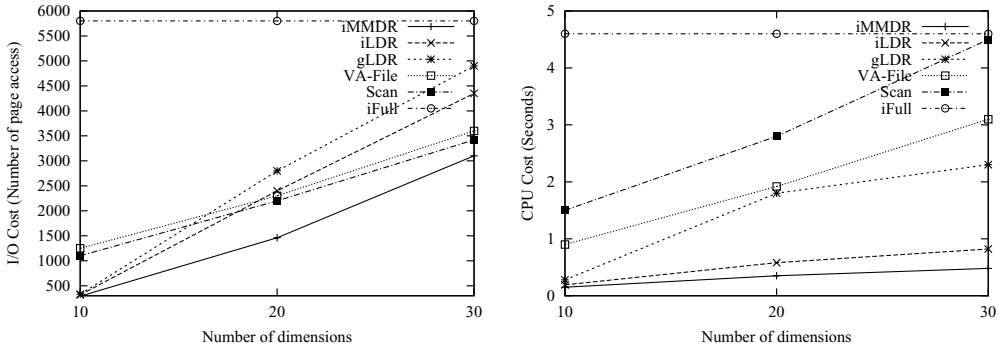
Fig. 15 Effect of dimensionality on query precision for wavelet feature

Figure 13 shows the effect of retained dimensionality on the query precision using the color histogram dataset. It is interesting that all five methods were not performing as well as before. Nevertheless, the MMDR method performed the best and was least affected. The higher precision obtained by the MMDR method confirms two important observations. First, there exist some local elongated clusters. Second, some intrinsic local elongated *shapes*/*correlations* cannot be detected by LDR. Compared to the synthetic dataset (Fig. 12), the precision of the methods on color histogram dataset is much worse. One reason could be that the real dataset might have clusters that are highly uncorrelated. Too many outliers might be another reason. This was possible, as for each image in the real dataset, the color histograms tended to be very skewed towards a small set of colors, with many attributes being 0. Figure 14 shows the effect of retained dimensionality on the query precision using the video frame's RGB color dataset. It has a similar performance to Fig. 13. Figure 15 shows the effect of retained dimensionality on the query precision using the wavelet feature dataset. It has a trend similar to Figs. 13 and 14, but with lower precisions. The reason is possibly that higher uncorrelated property exists in our WWW images. Since our images were randomly crawled from WWW, their wavelet features may not be well correlated in some clusters. However, the gap between LDR and MMDR became even larger for this dataset.

The aforementioned experiments confirm that the MMDR method is a much more effective dimensionality reduction technique in correlated environments with lower loss of distance information, as it can achieve better reduction performance with higher precision, which should lead to faster search and retrieval.

6.2 Query efficiency

In this experiment, we examined the query performance of the index methods on reduced dimensionality data points. Note that the final purpose of performing effective dimensionality reduction by using MMDR is to improve query performance, as it is well known that existing multidimensional indexing structures are not able to index very high dimensional (30-dimensional or higher) data space.



(a) on I/O Cost (b) on CPU Cost

Fig. 16 Effect of dimensionality on efficiency for small synthetic data

Here, we have five indexing schemes to compare: extended iDistance on MMDR data (iMMDR), extended iDistance on LDR data (iLDR), Global indexing method [4] on LDR data (gLDR) which is the state-of-the-art method in dimensionality reduction category, VA-File on MMDR data (VA-File) and sequential scan on MMDR data (Scan). The Global indexing method makes use of one Hybrid tree [21] for each cluster, and maintains the information about each cluster and index in an array. The VA-File is constructed by uniformly representing each dimension by 5 bits. Furthermore, to have a clearer picture on the effect of dimensionality reduction, we also present the performance of iDistance on the original data, i.e., full dimensional data (iFull). Here, we used the same datasets as in the previous section.

Figures 16–19 show the I/O and CPU cost of the five indexing schemes in reduced subspaces, when the subspace dimensionality varied from 10 to 30, and the iDistance on the full dimensional data. Figure 16a shows that for the synthetic dataset, as the dimensionality increased, the iMMDR had much lower I/O cost than the iLDR, which confirms that *a more effective dimensionality reduction method leads to an overall improved query efficiency*. We also notice that the gLDR was worse than the iLDR, which indicates that extended iDistance is more efficient than the Global indexing method. It is also obvious that the performances of VA-File and Scan on MMDR data are similar and worse than iMMDR. This is reasonable, since the synthetic data are skew/correlated and VA-File has been proved to be inefficient for skew data because of extensive random accesses [22]. This further confirms the superiority of the extended iDistance. The extended iDistance was more efficient in terms of I/O cost, as it had to traverse only one index, and this index was smaller since only one-dimensional distance values were used in the internal nodes. Figure 16a also indicates the performance of iDistance on the original data. As we can see, as less dimensions are retained, i.e., more dimensions are discarded, greater improvement can be achieved. However, recall that eliminating more dimensions results in lower precision, as shown in Fig. 12a. Hence, a trade-off between precision and performance is needed. For this dataset, 10–20 is a good range for the reduced dimensionality, since we can achieve more than 80% precision, while the I/O cost can be improved by nearly an order of magnitude.

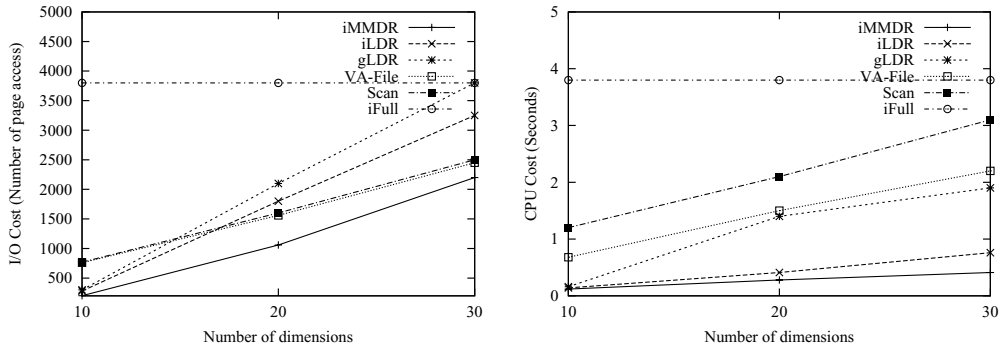
Figure 16b shows the CPU cost of the five indexing schemes for synthetic dataset and iFull. We can see that as the dimensionality increased, the gap became wider between iLDR and gLDR. iMMDR was the best. The performance difference between iMMDR and iLDR was relatively small. When the dimensionality reached 30, the CPU cost for gLDR was an order of magnitude higher than that for iMMDR and iLDR. The main reason is clear. In the gLDR indexing structure, tree nodes contain multi-dimensional data points. However, in the extended iDistance structure, tree nodes contain one-dimensional key values. Extended iDistances (iMMDR and iLDR) incur single-dimensional value comparison in searching, while L-norm computation is involved in the Hybrid-Tree. Thus, computation in gLDR is much more expensive. More L-norm computations occur in VA-File and sequential scan, since all approximations in VA-File and all data points in sequential scan are compared. As a results, both perform worst and get close to the CPU cost of iDistance on full dimensional data space.

Figures 17–19 show similar trends on I/O cost and CPU cost for the color histogram, wavelet feature, and video frame datasets respectively. Consequently, in terms of both I/O cost and CPU cost, the single dimensional extended iDistance index outperforms the Global indexing structure significantly. Furthermore, a more effective dimensionality reduction method leads to more efficient indexing.

6.3 Scalability

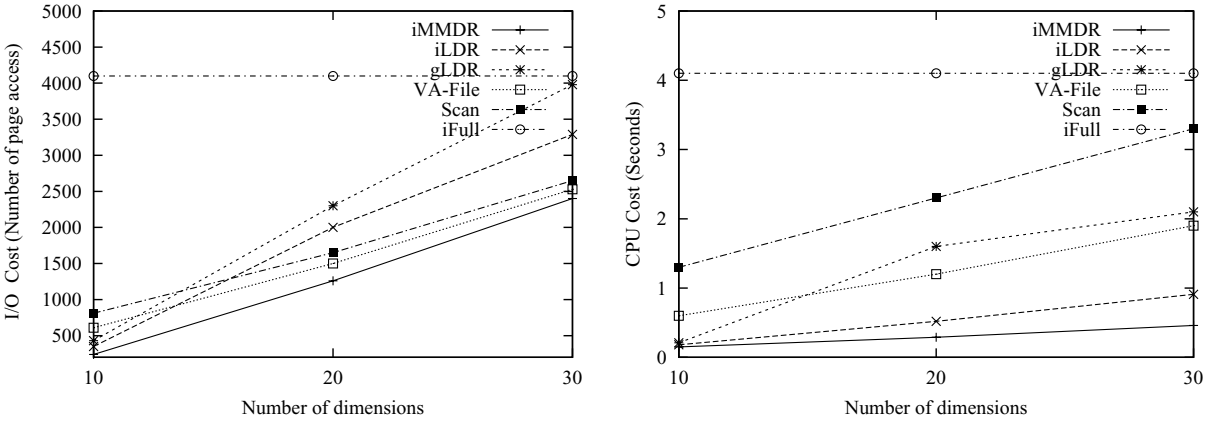
All high-dimensional indexes are affected by the data size and the number of dimensions of the data. In this experiment, we looked at the scalability of MMDR. We set the data stream ratio ε as 0.005, the k value in the lookup table to 3, and the number of iterations that indicated a point as $inactive$ as 10. The parameter we used here is the total response time (TRT) for MMDR to generate the optimal subspaces from the original data.

Figure 20a describes the effect of data size on the total response time. We kept the number of dimensions fixed at 100, while we varied the data size from 50,000 to 1,000,000.



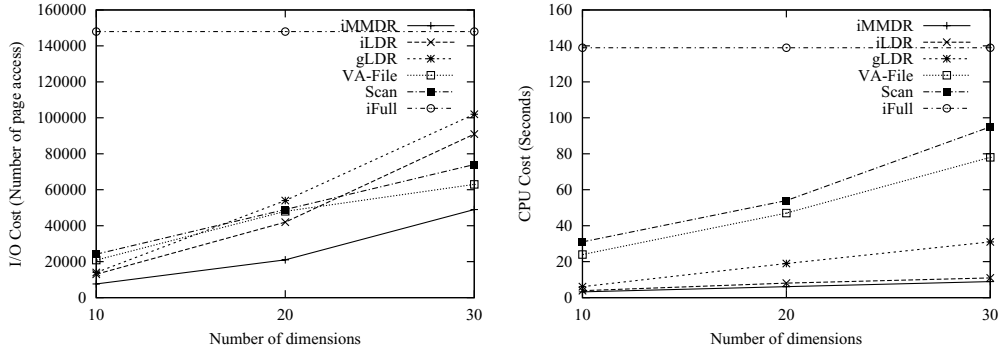
(a) on I/O Cost (b) on CPU Cost

Fig. 17 Effect of dimensionality on efficiency on color histogram



(a) on I/O Cost (b) on CPU Cost

Fig. 18 Effect of dimensionality on efficiency for wavelet feature



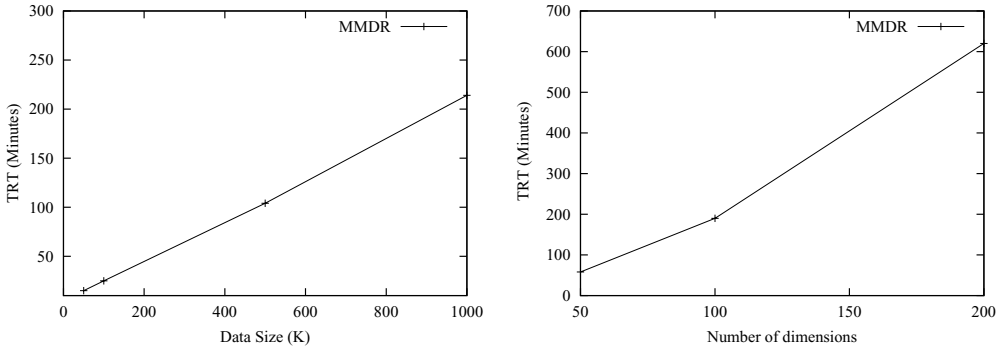
(a) on I/O Cost (b) on CPU Cost

Fig. 19 Effect of dimensionality on efficiency for video frame

From Fig. 20a, we make the following observation. The response time increased linearly to the data size. When the data size reached the limit of the buffer – 500K, there was no jump in response time for the Scalable MMDR, since we needed only to scan the whole dataset once. Figure 20b shows the effect of the number of dimensions on the total response time. For this experiment, we used 1,000,000 data points and varied the number of dimensions from 50 to 200. As expected, the total response time was nearly quadratic to the dimensionality. The results again exhibit that the limited buffer has no effect on the total response time.

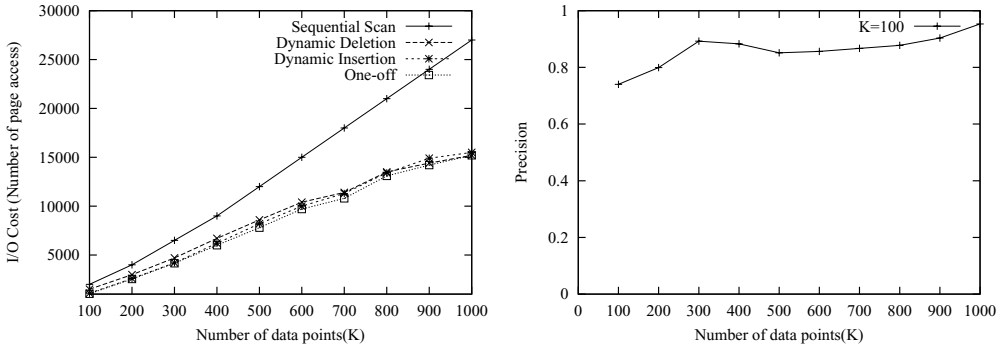
6.4 Effects of dynamic update

The dimensionality reduction algorithm and the associated indexing structure must be dynamic and adaptive in order to handle dynamic update effectively. In this experiment, we see how dynamic MMDR adapts and keeps its



(a) Effect of Data Size (b) Efect of Dimensionality

Fig. 20 Effect on total response time



(a) I/O Cost (b) Precision

Fig. 21 Effect on dynamic insertion

precision high, while keeping the KNN search cost low. For this experiment, we used the 1,000,000 100-dimensional dataset and 100 NN queries. For dynamic insertion, we constructed the index using the first 100,000 points and then we inserted 100,000 points at a time. At each insertion step, we performed KNN queries, and measured their query cost and precision. Meanwhile, we also study the performance of the dynamic deletion. For dynamic deletion, we constructed the index using the 1,000,000 points and then deleted 100,000 points at a time. The standard deletion operation in B^+-tree is performed. As a step further, we compare the dynamic insertion/deletion with one-off construction (i.e., index rebuilt on each insertion/deletion).

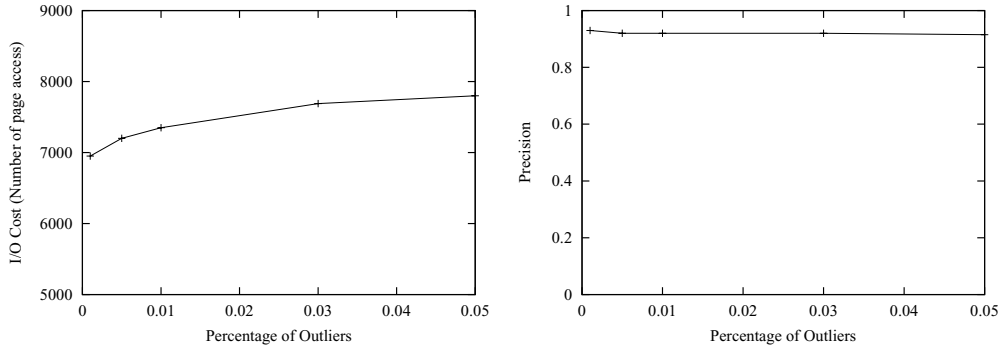
Compared to the performance of MMDR with one-off construction given in Fig. 21a, we notice that the performance of dynamic MMDR slightly degrades. For dynamic insertion, as more data points were inserted, a slightly higher I/O cost was incurred. Similarly, as more data points were deleted, a higher I/O cost happened too. However, the overhead is marginally small. This shows the robustness of our method to dynamic update, without rebuilding the index. Figure 21b shows the precision of MMDR as the data size varies. Note that given the same query and the same dataset on each insertion/deletion, the precision for dynamic insertion/deletion and one-off construction is the same, since an exact KNN search is performed for the query. As we can see from Fig. 21b, as the data size increased, MMDR adapted dynamically and yielded better precision. This again confirms the suitability of Mahalanobis distance in identifying effective clusters for dimensionality reduction for our method.

6.5 Effects of outliers

Outliers are data points that do not form effective clusters and are not included in any clusters obtained during dimensionality reduction. They form a cluster of their own, and they are indexed in their actual data space. Since outliers are composed of 'abnormal' points, the maximum radius of the cluster is very large after mapping into one-dimensional distance values. Hence, it is searched for most queries. We used the 1,000,000 100-dimensional dataset. Figure 22a shows that as more outliers were introduced, higher I/O cost was incurred. However, as shown in Fig. 22b, it is interesting that outliers did not affect query precision severely, since they were grouped into an outliers set and were searched like other clusters.

7 Conclusions

In this paper, we have presented an effective and dynamic dimensionality reduction algorithm – MMDR, which is able



(a) I/O cost (b) Precision

Fig. 22 Effect on outliers

Generate Correlated Dataset (GCD)

input: N,d,EC,EC_size[EC],s_dim[EC],s_r_dim[EC],
variance_e[EC],variance_r[EC], lb[EC]
Output: datasets[EC]
Algorithm:
1. for i from 0 to EC-1 do
2. for j from 0 to EC_size[i]-1 do
3. for k from 0 to s_remained_dim[i]-1
4. datasets[i][j*d+k]= gen_float(lb[i], variance_e[i])
5. for k from s_r_dim[i] to s_r_dim[i]+s_dim[i]-1
6. datasets[i][j*d+k]= gen_float(lb[i], variance_r[i])
7. for k from s_r_dim[i]+s_dim[i] to d-1
8. datasets[i][j*d+k]= gen_float(lb[i], variance_e[i])
9. rotate datasets[i] to be arbitrarily oriented

Fig. 23 Synthetic datasets generation

to reduce the number of dimensions while keeping the precision high, and is able to effectively handle large datasets and dynamic insertions. We used an extended iDistance to index the data points in different reduced subspaces. We conducted extensive experimental studies using both real and synthetic datasets to compare the algorithm with existing approaches. The results show that the proposed technique, as a whole, is very effective and efficient in supporting KNN search in very high-dimensional space. Furthermore, it is scalable for very large databases and able to handle dynamic insertions adaptively.

Acknowledgements We would like to thank Kaushik Chakrabarti for providing us with the source codes of LDR and Hybrid-tree. We also thank those anonymous reviewers for their valuable comments in improving the quality of our paper.

Appendix A: Generate synthetic datasets

In order to generate the local correlated datasets, we use the algorithm outlined in Fig. 23 to generate different clusters in different subspaces with different orientations and distensibilities retained. Table 2 gives the descriptions of input parameters.

In this algorithm, array *s_dim[i]* contains the dimensions that should be remained for each cluster. We can randomly choose which dimension should be retained. For simplicity, we make remained dimensions continuous starting with *s_r_dim[i]*. For example, if *s_r_dim[i]* = 6, then the remained dimensions for *i*th cluster starts from sixth to (6 + *s_dim)*th dimensions. Specifying the different values for each cluster allows each reduced subspace in different axis systems. Method $gen\_float()$ will return a random float value in $[lb, lb+variance]$. It can also return a value based on other distribution functions, such as Zipfian. For each cluster, we also specify their different lower bound values, which can be used to control the positions of centers of each cluster together with its variance. Along each of the remaining *s_dim[i]* dimensions, we assign a randomly chosen value falling in range of $[lb[i], lb[i]+variance\_r[i]]$ to all the points in the cluster. Along each of the reduced $(d - s\_dim[i])$ dimensions, we assign a randomly chosen value falling in range of $[lb[i], lb[i]+variance\_e[i]]$ to all the points in the cluster. The ratio between $variance\_r[i]$ and $variance\_e[i]$ in fact specifies the ratio between the energy carried by remained and reduced dimensions for each cluster, or the degree of correlation/ellipticity. Both values can be adjusted for different clusters in order to have different level of correlation. To make the subspace arbitrarily oriented, we can generate a random orthogonal rotation matrix (generated using MATLAB) and rotate the cluster by multiplying the data matrix with the rotation matrix.

Table 2 Table of input parameters and description

Parameters	*Descriptions*
N	Data size
d	Original dimensionality
s_dim	Subspace dimensionality
EC	Number of elliptical clusters
EC_size	Size of elliptical cluster
s_r_dim	Starting remained dimension for EC
$variance\_e$	Variance for eliminated subspace
$variance\_r$	Variance for remained subspace
lb	Lower bound value for EC

References

1. Yu, C.: High-dimensional indexing. Ph.D. thesis, Department of Computer Science, National University of Singapore (2001)
2. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is nearest neighbors meaningful? In: ICDT, pp. 217–235 (1999)
3. Ooi, B.C., Tan, K.L., Yu, C., Bressan, S.: Indexing the edges: a simple and yet efficient approach to high-dimensional indexing. In: PODS, pp. 166–174 (2000)

4. Chakrabarti, K., Mehrotra, S.: Local dimensionality reduction: a new approach to indexing high dimensional spaces. In: VLDB, pp. 89–100 (2000)
5. Jolliffe, I.T.: Principal Component Analysis. Springer-Verlag, Berlin Heidelberg New York (1986)
6. Yu, C., Ooi, B.C., Tan, K.L., Jagadish, H.V.: Indexing the distance: an efficient method to KNN processing. In: VLDB, pp. 166–174 (2001)
7. Jin, H., Ooi, B.C., Shen, H.T., Yu, C., Zhou, A.: An adaptive and efficient dimensionality reduction algorithm for high-dimensional indexing. In: ICDE, pp. 87–98 (2003)
8. Böhm, C., Berchtold, S., Keim, D.: Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. ACM Comput. Surv. **33**(3), 322–373 (2001)
9. Weber, R., Schek, H., Blott, S.: A quantitative analysis and performance study for similarity search methods in high dimensional spaces. In: VLDB, pp. 194–205 (1998)
10. Berchtold, S., Böhm, C., Kriegel, H.-P.: The pyramid-technique: towards breaking the curse of dimensionality. In: SIGMOD, pp. 142–153 (1998)
11. Aggarwal, C.C., Wolf, J.L., Yu, P.S., Procopiuc, C., Park, J.S.: Fast algorithms for projected clustering. In: SIGMOD, pp. 61–72 (1999)
12. Hinneburg, A., Keim, D.A.: An optimal grid-clustering: towards breaking the curse of diminsionality in high dimensional clustering. In: VLDB, pp. 506–517 (1999)
13. Vitter, J.S., Wang, M.: Approximate computation of multidimensional aggregates of sparse data using wavelets. In: SIGMOD, pp. 193–204 (1999)
14. Lee, J.H., Kim, D.H., Chung, C.W.: Multi-dimensional selectivity estimation using compressed histogram information. In:SIGMOD, pp. 205–214 (1999)
15. Aggarwal, C.C., Hinneburg, A., Keim, D.A.: On the surprising behavior of distance metrics in high dimensional spaces. In: ICDT, pp. 420–434 (2001)
16. Duda, R.: Pattern recognition for HCI. http://www.engr.sjsu.edu/k̃napp/
17. Sung, K.K., Poggio, T.: Example-based learning for view-based human face detection. In: PAMI **20**(1), 39–51 (1998)
18. Wang, J.Z., Wiederhold, G., Firschein, O., Wei, S.X.: Content-based image indexing and searching using daubechies wavelets. In. J. Digital Lib. **1**(4), 311–328 (1998)
19. http://www.virtualdub.org
20. Wu, Y.-L., Agrawal, D., Abbadi, A.E.: A comparison of DFT and DWT based similarity search in time-series databases. In: CIKM, pp. 488–495 (2000)
21. Chakrabarti, K., Mehrotra, S.: The hybrid tree: an index structure for high dimensional feature spaces. In: ICDE, pp. 322–331 (1999)
22. Sakurai, Y., Yoshikawa, M., Uemura, S., Kojima, H.: The A-tree: an index structure for high-dimensional spaces using relative approximation. In: VLDB, pp. 516–526 (2000)