Towards Effective Indexing for Very Large Video Sequence Database

Heng Tao Shen
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane QLD 4072 Australia
shenht@itee.uq.edu.au

Beng Chin Ooi
Department of Computer Science
National University of Singapore
Kent Ridge, Singapore 117543
ooibc@comp.nus.edu.sg

Xiaofang Zhou
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane QLD 4072 Australia
zxf@itee.uq.edu.au

ABSTRACT

With rapid advances in video processing technologies and ever fast increments in network bandwidth, the popularity of video content publishing and sharing has made similarity search an indispensable operation to retrieve videos of user interests. The video similarity is usually measured by the percentage of similar frames shared by two video sequences, and each frame is typically represented as a high-dimensional feature vector. Unfortunately, high complexity of video content has posed the following major challenges for fast retrieval: (a) effective and compact video representations, (b) efficient similarity measurements, and (c) efficient indexing on the compact representations. In this paper, we propose a number of methods to achieve fast similarity search for very large video database. First, each video sequence is summarized into a small number of clusters, each of which contains similar frames and is represented by a novel compact model called Video Triplet (ViTri). ViTri models a cluster as a tightly bounded hypersphere described by its *position*, *radius*, and *density*. The ViTri similarity is measured by the volume of intersection between two hyperspheres multiplying the minimal density, i.e., the estimated number of similar frames shared by two clusters. The total number of similar frames is then estimated to derive the overall similarity between two video sequences. Hence the time complexity of video similarity measure can be reduced greatly. To further reduce the number of similarity computations on ViTris, we introduce a new one dimensional transformation technique which rotates and shifts the original axis system using PCA in such a way that the original inter-distance between two high-dimensional vectors can be maximally retained after mapping. An efficient B^+-tree is then built on the transformed one dimensional values of ViTris' positions. Such a transformation enables B^+-tree to achieve its optimal performance by quickly filtering a large portion of non-similar ViTris. Our extensive experiments on real large video datasets prove the effectiveness of our proposals that outperform existing methods significantly.

SIGMOD 2005 June 14-16, 2005, Baltimore, Maryland,USA

1. INTRODUCTION

In recent decades, multimedia retrieval has attracted plenty of attention due to its data's rich content. Among media types, video presents the most complex data, including a sequence of frames (or feature vectors), audio, motion, etc. With ever more heavy usage of video devices and advances in video processing technologies, the amount of video data has grown rapidly and enormously for various usages, such as advertising, news video broadcasting, personal video archive, medical video data, and so on. Interestingly, the popularity of WWW enables enormous video data to be published and shared. Web search engines provide users convenient ways for finding videos of their interests. Due to the high complexity of video data, retrieving the similar video content with respect to a user's query from a large database requires: (a) effective and compact video representations, (b) efficient similarity measurement, and (c) efficient indexing on the compact representations.

Here we define a video as a sequence of frames, each of which is a high-dimensional image feature vector. The number of frames is typically in the range of hundreds or more, depending on the length of video. Generally, similarity between two video sequences is measured by identifying the closest match for every single frame in each sequence. The time complexity is linear to the product of the lengths of the two sequences [6]. Obviously, such a process is impractical for very large video databases. One promising way to reduce the computational cost is to construct a compact summary which is described by much fewer keyframes/representatives. The video similarity is then approximated based on the summary similarity. Unfortunately, existing techniques all suffer from the drawback of losing information localized to each individual representative in the summary [13, 8, 14]. This obviously limits the retrieval accuracy.

In this paper, we introduce a novel summary representation model called Video Triplet (ViTri). A sequence is first summarized into a small number of clusters which contain similar frames. Each cluster is then modelled as a hypersphere in n-dimensional space, where n is the dimensionality of feature vector. Each hypersphere is represented by a video triplet: (position, radius, density), which determine

the cluster center, cluster range, and number of frames in the cluster respectively. Particularly, we refine the cluster radius by mean and standard deviation of the distances between frames to the cluster center. Unlike existing methods that measure the similarity of two clusters by the distance between two cluster centers, we evaluate their similarity by the estimated number of similar frames shared by two clusters, which is computed by the volume of intersection between two hyperspheres multiplied by the smaller density. By doing so, the local information for each cluster can be captured more accurately. Finally the total number of similar frames shared by two video sequences can be computed efficiently. Hence the time complexity of similar measure is reduced to be linear to the product of number of clusters in two sequences.

For a database containing a very large number of video sequences, exhaustive search among all ViTris to identify the similar videos are strongly undesirable. To mitigate this problem for further improvement of retrieval response, one strategy is to design an efficient indexing method to avoid extensive accesses and redundant computation on the non-similar ViTris. Inspired by the simplicity and efficiency of B^+-tree, we propose a novel one dimensional transformation method to map a n-dimensional feature vector into a one dimensional distance value. By doing so, the positions of ViTris can be indexed by an efficient B^+-tree. We rotate and shift the original axis system by using Principle Component Analysis (PCA) to find an optimal reference point such that the original inter-distance between two high-dimensional feature vectors can be maximally retained after the transformation. Given a search radius, fewer ViTris are then expected to be accessed, so as to the number of similarity computations.

Our extensive performance study on large real life video datasets indicate that the proposed ViTri method offers better accuracy than existing video summarization methods and our one dimensional transformation achieves significant performance advantages over sequential scan and existing indexing methods.

The rest of paper is organized as follows. In Section 2, we review related work. Foundations of our work is presented in Section 3, followed by ViTri and its measure in Section 4. We present the method to index ViTris in Section 5. An extensive performance study is reported in Section 6, and finally, we conclude our paper in Section 7.

2. RELATED WORK

In content-based image retrieval, the similarity/distance of two images is typically computed by the well-known Euclidean distance. When extending the distance function to video sequences, many proposals have been proposed. One widely used video similarity measure is the percentage of similar frames shared by two sequences [6]. Warping distance [13] is used to measure the temporal differences between two sequences. Hausdorff distance [5] is used to measure the maximal dissimilarity between two shots. Template matching of shot-change duration [7] is used to compute the overlap between two sequences. However, all these measures need to compare most, if not all, frames pairwise. The full similarity computation requires storage of the entire sequence and time complexity is at least linear to the number of frames multiplied by the dimensionality of frame. Obviously, such expensive compactions are strongly prohibited for very large video datasets.

To reduce the computational cost, one approach is to summarize the video sequences into compact representations, hence the video similarity is approximated based on the compact representation. In literature, two categories of summarization techniques have been proposed: summarize the sequence as a statistical distribution and summarize the sequence into fewer representatives. The first category typically assumes that the frames are distributed in a model like Gaussian, or mixture of Gaussian [8, 14]. In the second category, [5] identifies keyframes by selecting the k feature vectors to minimize the distance between them and the original sequence. [6] introduces a randomized summarization method which randomly selects a number of seed frames and assigns a small collection of closest frames (called video signature) to each seed. However, the selection of seeds may sample non-similar frames from two almost-identical sequences. Furthermore, too many parameters need to be tuned for better performance.

High-dimensional indexing is another topic related to our research. High-dimensional indexing [3] have been extensively researched in the database literature. Our work is more closely related to techniques based on transforming high-dimensional data to single dimensional space. The Pyramid technique [2] and the iDistance scheme [15] are two typical examples. The Pyramid technique divides the D-dimensional data space into 2D 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 transforms a high-dimensional point into a single-dimensional distance value with reference to its corresponding reference point.

3. FOUNDATIONS

In this section, we provide the basic definitions that serve as the foundations of our work.

3.1 Video Similarity Measure

We assume each frame in a video sequence is represented by a high-dimensional feature vector (hence frame and feature vector are interchangeable). A video sequence X is then represented as: $X = \{x_1, x_2, ..., x_f\}$ where f is the number of frames in the sequence. d(x,y) is the distance function to measure the similarity/dissimilarity of two frames x and y. We say x and y are similar if $d(x, y) <= \epsilon$, where ϵ is a small positive value and called frame similarity threshold. ϵ may depend on different feature spaces and is tuned for precision-efficiency trade-off in retrieval performance. The similarity between two sequences is then defined as [6]:

$$sim(X, Y) = \frac{\sum_{x \in X} 1_{\{y \in Y : d(x,y) \leq \varepsilon\}} + \sum_{y \in Y} 1_{\{x \in X : d(x,y) \leq \varepsilon\}}}{|X| + |Y|}$$

Where $|\cdot|$ is the number of frames in the sequence. This similarity is measured by the percentage of similar frames in two sequences and robust to temporal order of frames. In this paper, we use this definition to measure the video similarity. Obviously, as mentioned, the time complexity of this measure is too high to for practical usage. One of our goals in this paper is to discover an efficient method to estimate the number of similar frames precisely, without computing the inter-similarity of frames.

3.2 Volumes

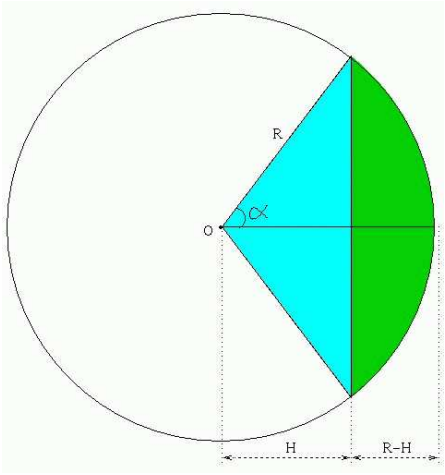


Figure 1: A 2-dimensional example.

In a n-dimensional feature space, hypersphere (often simply called the n-sphere) is a generalization of the circle in 2-dimensional space and usual sphere in 3-dimensional space to dimensions. A hypersphere is therefore defined as a set of n-dimensional vectors whose distances to the center are less or equal to the radius. Similarly, hypersector, hypercone, and hypercap are generalizations of the sector, cone and cap respectively. As a 2-dimensional example shown in Figure 1, the circle is determined by center O and radius R. The sector inside of the circle, the cone inside of the sector (light shaded area) and the cap inside of sector (dark shaded area) are all determined by angle α since the heights of cone and cap can be derived from α.

It is known that the volume of a n-dimensional hypersphere centered at O with radius of R, denoted as V(O,R), is computed as following:

$$V_{hypersphere}(O,R) = \frac{\pi^{\frac{n}{2}}}{\Gamma(1+\frac{n}{2})}R^n = \begin{cases} \frac{\pi^{\frac{n}{2}}}{(\frac{n}{2})!}R^n \\ \frac{2^{n+1}\pi^{\frac{n-1}{2}}(\frac{n+1}{2})!}{(n+1)!}R^n \end{cases}$$

In the above and following formulas on volume, the upper formula inside of bracket is for even n and the lower one is for odd n.

Similarly by using integral, the volume of hypersector is computed as [9]:

$$V_{hypersector}(O,R,\alpha) = R^n \frac{2\pi^{\frac{n-1}{2}}}{n\Gamma(\frac{n-1}{2})}\int_0^\alpha (\sin(\alpha))^{n-2}d\alpha = \begin{cases} R^n \frac{\pi^{\frac{n-2}{2}}}{(\frac{n}{2})!}\left(\alpha - \cos(\alpha)\sum_{i=0}^{\frac{n-4}{2}}\frac{2^{2i}(i!)^2}{(2i+1)!}(\sin(\alpha))^{2i+1}\right) \\ R^n \frac{2^n\pi^{\frac{n-1}{2}}(\frac{n+1}{2})}{(n+1)!}\left(1 - \cos(\alpha)\sum_{i=0}^{\frac{n-3}{2}}\frac{(2i!)^2}{2^{2i}(i!)^2}(\sin(\alpha))^{2i}\right) \end{cases}$$

And the volume of Hypercone is:

$$V_{hypercone}(O,R,\alpha) = R^n \frac{\pi^{\frac{n-1}{2}}}{n\Gamma(\frac{n+1}{2})}\cos(\alpha)(\sin(\alpha))^{n-1} = \begin{cases} R^n \frac{2^{n-1}\pi^{\frac{n-2}{2}}}{n!}\cos(\alpha)(\sin(\alpha))^{n-1} \\ R^n \frac{\pi^{\frac{n-1}{2}}}{n(\frac{n-1}{2})!}\cos(\alpha)(\sin(\alpha))^{n-1} \end{cases}$$

The volume of hypercap is simply derived by deducting the volume of hypercone from that of hypersector:

$$V_{hypercap}(O,R,\alpha) = V_{hypersector}(O,R,\alpha) - V_{hypercone}(O,R,\alpha) = \begin{cases} R^n \frac{\pi^{\frac{n-2}{2}}}{(\frac{n}{2})!}\left(\alpha - \cos(\alpha)\sum_{i=0}^{\frac{n-2}{2}}\frac{2^{2i}(i!)^2}{(2i+1)!}(\sin(\alpha))^{2i+1}\right) \\ R^n \frac{2^n\pi^{\frac{n-1}{2}}(\frac{n+1}{2})}{(n+1)!}\left(1 - \cos(\alpha)\sum_{i=0}^{\frac{n-1}{2}}\frac{(2i!)^2}{2^{2i}(i!)^2}(\sin(\alpha))^{2i}\right) \end{cases}$$

Note that the term coming from the formula of the hypercone constitutes an additional term which supplements the series. The formula is identical to that of the hypersector, except the number appearing in the top of sigma [9].

The above formulas pave the way to estimate the number of similar frames shared by two clusters which are modelled as ViTris in Section 4.

3.3 Principal Component Analysis

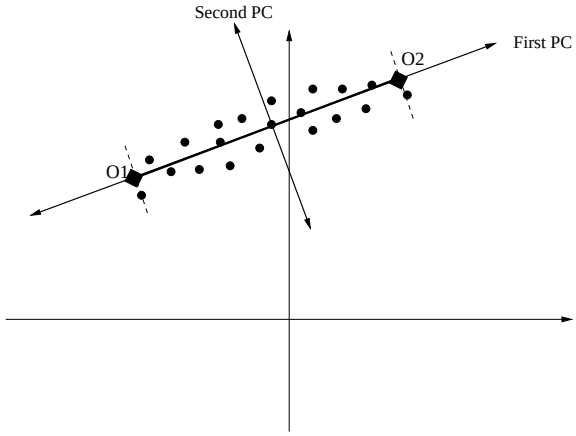


Figure 2: PCs in 2-dimensional space.

Principal Component Analysis (PCA) [11] 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** and exhibits the largest variance. The second component corresponds to the eigenvector with the second largest eigenvalue and exhibits the second largest variance, and so on. All principal components are orthogonal to each other.

A 2-dimensional example is shown in Figure 2, where the first principal component indicates the direction that exhibits the largest variance, and the second principal component that is orthogonal to the first one indicates the direction with the smallest variance. Next, we present a useful definition called variance segment.

DEFINITION 1 (VARIANCE SEGMENT). *For a principle component, denoted as Φ_i which identifies a line (or a direction), its variance segment is a segment of the line determined by two furthermost projections on Φ_i for all data points.*

In the example shown in Figure 2, on the line identified by the first principle component Φ_1, projections of O_1 and O_2, computed as $O_1 \cdot \Phi_1$ and $O_2 \cdot \Phi_1$ (indicated as diamonds in the figure), are two furthermost projections. The segment between $O_1 \cdot \Phi_1$ and $O_2 \cdot \Phi_1$ is the variance segment for Φ_1 (indicated as bold in the figure). The definition of variance segment will help us in finding an optimal reference point for one dimensional transformation.

4. VIDEO TRIPLET

In this section, we see how the time complexity of similar measure between two videos can be reduced. We first present how to generate clusters of similar frames, followed by its representation - ViTri. We then discuss how to measure the similarity of two ViTris, from which the similarity of two videos is also derived.

4.1 ViTri Representation

The video similarity measure in this paper models a video sequence as a collection of frames and is robust to temporal ordering. Observing that nearby frames in a video are quite similar (for example, frames in a shot), we summarize a video sequence into a small number of clusters which contain similar frames, and one frame belongs to one and only one cluster. An ideal cluster should satisfy two conditions: contains similar frames only and no other frame similar to the frames in the cluster can be found in any other cluster. However, such an ideal cluster may not exist for any arbitrary sequence. Instead, we relax the cluster to only contain similar frames, i.e., the similarity of any two frames in a cluster should not greater than ϵ.

To generate clusters of similar frames, we deploy the well-known k-means method to construct a recursively binary clustering method as shown in Figure 3.

Generate_Clusters(X)
1. clusters[] ← k-means(X,2);
2. for each cluster C centered at O
3. compute μ and σ of $d(x_i, O)$ $\forall x_i \in C$;
4. if R=min(R, $\mu+\sigma$) $\leq \epsilon/2$
5. record O, R, and $|C|$;
6. else
7. Generate_Clusters(C_i);

Figure 3: Clusters Generation Algorithm.

The algorithm is initialized by a video sequence X. It first generates two clusters based on k-means method (line 1). For each cluster, we compute the mean, denoted as μ, and the standard deviation of mean, denoted as σ, of the distances of all frames with respective to the cluster center (line 3). The cluster radius R is refined as min(R, $\mu+\sigma$). If R is not greater than $\epsilon/2$, we treat the cluster as a valid cluster and record its center, radius, and size (i.e., number of frames) of the cluster (line 4-5). Otherwise, we recursively call Generate_Clusters by passing in the cluster to get smaller and more compact clusters (line 6-7).

The key of the above algorithm is the refinement of cluster radius. To enforce the condition that all frames in a cluster are similar, a safe requirement is to enforce the cluster radius to be less than $\epsilon/2$, i.e., R $\leq \epsilon/2$. However, in our algorithm, we refine the radius to be min(R, $\mu+\sigma$) to get a tighter cluster for the following reasons. First, from the formula on volume of hypersphere, a *small* enlargement in radius will result in a *significant* enlargement on the volume. For instance, increasing the radius by 10%, the volume of a 64-dimensional hypersphere will be increased by about 445 times (1.1^{64}). In real situations, it is not uncommon that there are few extreme values that are far away from the mean. Eliminating the effect of those extremes will tighten the space of a cluster and obviously result in a more compact representation. Second, statistically, most frames have distances less than or equal to $\mu+\sigma$ with respect to the center. σ is a measure of the spread of the distances from their mean and defined as:

$$\sigma = \sqrt{\frac{\sum (d(x_i, O) - \mu)^2}{|C|}}$$

The more spread apart the distances are, the higher the deviation. Assume the distances values are normally distributed, then 68% of frames have distances between (μ-σ, $\mu+\sigma$). Thus it is about 84% (i.e., 68%+32/2%) of frames have distances less than or equal to $\mu+\sigma$ from the cluster center. For a 64-dimensional hypersphere, increasing $\mu+\sigma$ by 10% means that only 16% of frames are distributed in the enlarged (by 445 times) volume. Third, for normally distributed hypersphere, it has also been proven that $\mu+\sigma$ is very close to the radius [12]. In short, based on the above analysis, in this paper we use min(R, $\mu+\sigma$) as cluster C's radius.

Obviously, the value of ϵ will affect the number of clusters generated for a video sequence. The smaller the ϵ is, the larger number of clusters; vice versa. When ϵ approaches to 0, each frame becomes a cluster. The degree of clustering is 0 and the best precision is expected. On the other hand, when ϵ tends to be large enough, all frames form a single cluster. The degree of clustering is maximized and the precision is expected to be low. Hence varying ϵ achieves precision-efficiency tradeoff, as we will see in experiments. The time complexity of the clusters generation algorithm is ln($|C|$) times the complexity of k-means. This is a preprocessing step for indexing.

After we get the clusters of a video sequence, next is to represent each cluster in a meaningful way such that effective and fast similarity measurement can be made. To do so, we introduce Video Triplet (ViTri).

DEFINITION 2 (VIDEO TRIPLET). *Video Triplet is a triplet of (position, radius, density). Position is the cluster center (denoted as O) which indicates the position of the cluster in the space. Radius (denoted as R) suggests the volume of a hypersphere which bounds the cluster. Density (denoted as D) is measured by the number of frame in the cluster C divided by its volume, i.e.,*

$$D = \frac{|C|}{V_{hypersphere}(O, R)}$$

□

Clearly, ViTri(O,R,D) describes several properties of a cluster. Unlike existing methods that only use the posi-

tion, we also investigate the properties on the volume and density for more accurate representation.

So far, a video sequence has been summarized into a set of clusters that are represented by ViTris. Next, we look at how the similarity between two ViTris is computed.

4.2 Similarity Measure

Analogous to the similarity measure of two videos, we also measure the similarity of two ViTris by the number of similar frames, which is estimated by the volume of intersection between two ViTris multiplied by the minimal density. Given two ViTris: $ViTri_1(O_1, R_1, D_1)$ and $ViTri_2(O_2, R_2, D_2)$, their similarity is measured subjective to the following cases. We assume $R_1 > R_2$ for easy illustration and d is the distance function used.

1. $d(O_1, O_2) \geq R_1 + R_2$:

   In this case, there is no intersection between two hypersphere identified by (O_1, R_1) and (O_2, R_2) as shown in Figure 4. Hence $sim(ViTri_1, ViTri_2)$=0.

2. $R_2 \leq d(O_1, O_2) < R_1 + R_2$:

   In this case, the value of β is less than or equal to $\frac{\pi}{2}$ as shown in Figure 5. The volume of intersection between two hypersphere is the sum of two hypercaps identified by (O_1, R_1, α) and (O_2, R_2, β). That is

   $sim(ViTri_1, ViTri_2)$= $(V_{hypercap}(O_1, R_2, \alpha)+V_{hypercap}(O_2, R_2, \beta))$*min$(D_1, D_2)$

3. $R_1 - R_2 \leq d(O_1, O_2) < R_2$:

   In this case, the value of β is less than or equal to $\frac{\pi}{2}$ as shown in Figure 6. Notice that the β here is equal to π - β of Case 2. The intersection of two hyperspheres consists of 2 parts. The first part is the hypercap identified by (O_1, R_1, α). The second part is the volume of hypersphere identified by (O_2, R_2) deducted by the volume of hypercap identified by (O_2, R_2, β). That is

   $sim(ViTri_1, ViTri_2)$= $(V_{hypercap}(O_1, R_1, \alpha) + V_{hypersphere}(O_2, R_2) - V_{hypercap}(O_2, R_2, \beta))$*min$(D_1, D_2)$

4. $d(O_1, O_2) < R_1 - R_2$:

   In the last case, the smaller hypersphere is completely contained in the larger one as shown in Figure 7. Straightforwardly,

   $sim(ViTri_1, ViTri_2)= V_{hypersphere}(O_2, R_2)$*min$(D_1, D_2)$

Given d(O_1, O_2), R_1 and R_2 which form a triangle, α and β can be derived easily by the law of sine or cosine. The above similarity measure involves the computations on volumes of hypersphere and hypercap. Recall the formulas on hypersphere and hypercap. Since the dimensionality of data space n is known, all the coefficients of R^n and sin(α) become constants and can be pre-computed. Consequently, in the worst case (Case 2 and 3), the complexity of similarity measure on ViTri is on the sum of $\frac{n}{2}$ terms in computing the volume of hypercap. However, the standard Euclidean distance even requires sum of n terms. Clearly, our similarity measure on ViTri is efficient.

After computing the inter-similarity between any pair of ViTris from two video sequences, the total number of similar frames is then calculated to compute the final similarity of

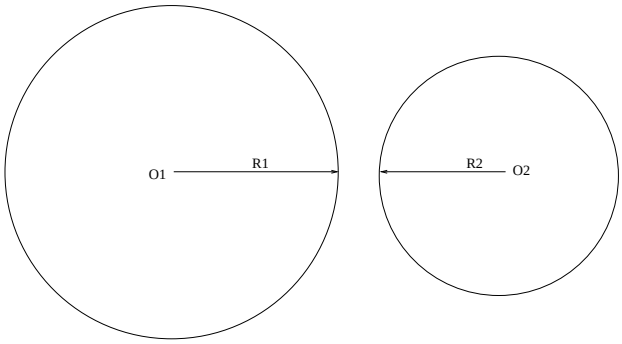


Figure 4: $d(O_1, O_2) \geq R_1 + R_2$

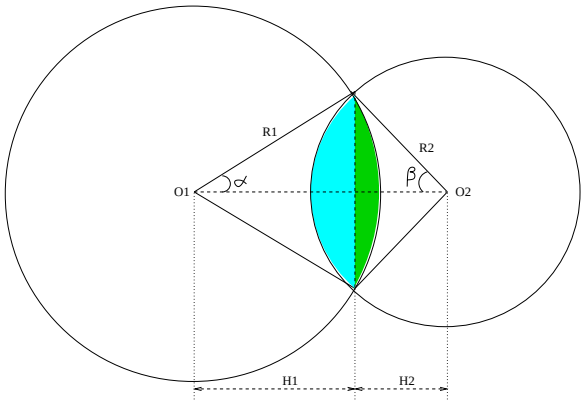


Figure 5: $R_2 \leq d(O_1, O_2) < R_1 + R_2$

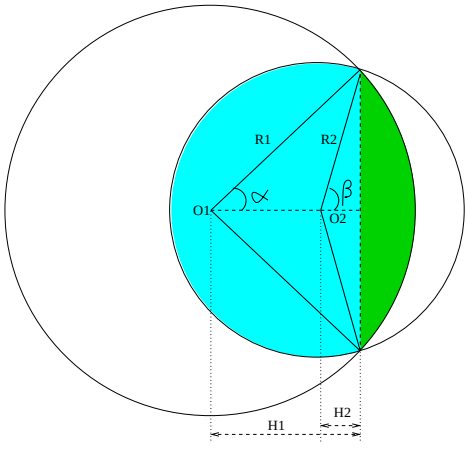


Figure 6: $R_1 - R_2 \leq d(O_1, O_2) < R_2$

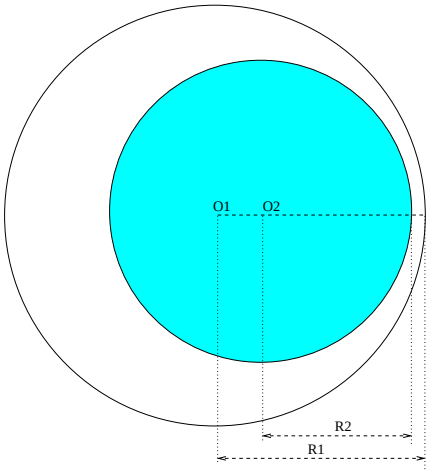


Figure 7: $d(O_1, O_2) < R_1 - R_2$

Notation	Description
ViTri	Video Triplet
O	Cluster Center
R	Cluster Radius
D	Cluster Density
ϵ	Frame Similarity Threshold
V	Volume
n	Feature Space Dimensionality
d	Distance Function
γ	Search Radius
O'	Reference Point
Φ	Principle Component

Table 1: A Table of Notations.

two videos. One distinct feature of our measure is that the final similarity of two videos is still estimated based on the number of similar frames, instead of the number of similar clusters [6, 5, 7, 13].

So far, we have reduced the time complexity of measuring two video sequences to be linear to the product of numbers of ViTris from the numbers of frames. ViTris are stored in the database to facilitate the retrieval. For a very large database containing tens of thousands of video sequences, exhaustive search among millions of ViTris to identify the similar videos are still strongly prohibitive. Case 1 suggests that some ViTris are indeed not similar to each other, i.e., share none of similar frames. This provides large room for further reducing the response time by only accessing and comparing a small portion of ViTris in the database. Indexing is a primary method to utilize the data access. In the next Section, we present how the ViTris are indexed to avoid a large amount of non-similar ViTris from being accessed and compared. For easy reference, a list of notations is shown in Table 1.

5. INDEXING VITRIS

In previous section, we have discussed how the time complexity of similarity measure between two videos can be reduced effectively. In this section, we focus on how to reduce the number of page accesses and similarity computations on ViTris.

5.1 One Dimensional Transformation

The B^+-tree has been used widely for many applications in various domains for its known simplicity and efficiency. Transforming high-dimensional feature vectors into one dimensional values which are indexed by a B^+-tree has shown its effectiveness recently [15, 10].

Generally, the design of one dimensional mapping was motivated by two factors. First, high-dimensional feature vectors can be ordered based on their distances to a reference point, and indexed based on such distance value. This enables one to represent high-dimensional data in a single dimensional space and use an existing B^+-tree. Second, the triangular inequality relationships enable the (dis)similarity between a query vector and a database vector to be derived with reference to the chosen reference point.

In [15], the one dimensional transformation for a data space is simply achieved by the following mapping function:

$$key = d(O_i, O')$$

where O_i is a high-dimensional vector (or interchangeably point), O' is the chosen reference point and d is the distance function to compute the distance between O_i and O'. Key is the derived one dimensional distance value for O_i. The keys of all points in the data space are then ordered and indexed by a B^+-tree. Given a search radius γ and a query Q, a range search $[d(Q, O')\text{-}\gamma,\ d(Q, O')\text{+}\gamma]$ is performed. Those points whose keys (i.e., distances to the reference point) are greater than $d(Q, O')+\gamma$ or less than $d(Q, O')-\gamma$ can be safely pruned by the triangular inequality. Figure 8 depicts a 2-dimensional example where only those points lying in the shaded area are accessed. Obviously, such a one dimensional mapping is information lossy in nature, i.e., far way points in original space may have the same keys. Look at Figure 8 again, points on the circle have the same keys.

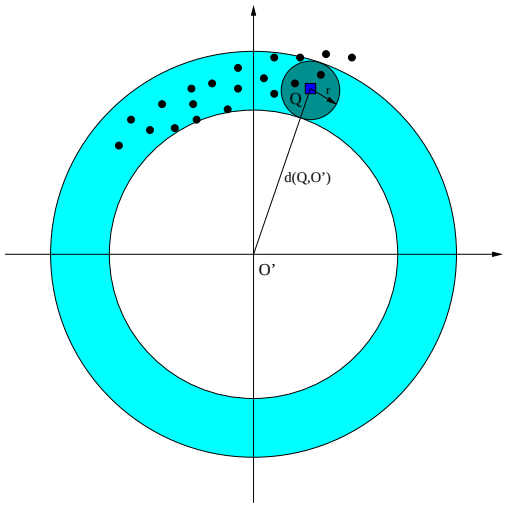


Figure 8: Points accessed by using space center as reference point

Notice that in B^+-tree, if keys are evenly distributed, then given a search radius, the larger the range of keys is, the less portion of keys are compared. Hence the challenge now is how to find a reference point such that the *variance* of inter-distances of points in the transformed one dimensional space

can be maximized. The distance of two points in original space is $d(O_i, O_j)$. After being transformed into one dimensional space with respective to O', the distance of two points becomes $\|d(O_i, O') - d(O_j, O')\|$ which is the distance information remained. $d(O_i, O_j)$-$\|d(O_i, O') - d(O_j, O')\|$ is the distance information lost. An optimal reference point is a point that maximizes the variance of $\|d(O_i, O') - d(O_j, O')\|$. Formally, we define the optimal reference point as follows:

DEFINITION 3 (OPTIMAL REFERENCE POINT). *Given a set of N n-dimensional points, denoted as $(O_1, O_2, ..., O_N)$, and the transformation function of d, an optimal reference point Q' is a point maximizing the variance of the inter-distances of points in transformed one dimensional space, where the inter-distances of points in the transformed space is $\|d(O_i, O') - d(O_j, O')\|$, $1 \leq i, j \leq N$.*

□

An optimal reference point aims to minimize the inter-distance information lost caused by transformation. Transformation by an optimal reference point leads to the maximal variance of keys, hence to more effective indexing by B^+-tree. [15] investigated effects of various choices of reference points. In this paper, we aim to find an optimal reference point to improve the performance. It is obviously impossible to test very position in the space for finding an optimal reference point. Fortunately, PCA can be used to find the direction of largest variance for a dataset [11, 4], as discussed in Section 3.3. By applying PCA, we have the following theorem on the positions of optimal reference points.

THEOREM 1. *Given a set of N n-dimensional points, denoted as $(O_1, O_2, ..., O_N)$, and the transformation function of d, the optimal reference points lie on the line identified by the first principle component Φ_1 and out of Φ_1's variance segment.*

Proof: *In PCA, the variance along a principle component Φ_i for a dataset is measured by the inter-distances of points along Φ_i. We first prove that a point O' lying on the line identified by Φ_i and out of Φ_i's variance segment preserves the variance of Φ_i.*

We represent Φ_i's variance by the distance of two points O_1 and O_2 on Φ_i as shown in Figure 9. Based on the triangle inequality, we have $d(O_1, O_2) \geq \|d(O_1, O') - d(O_2, O')\|$. Since O' lie on Φ_i and is out of the segment identified by O_1 and O_2. The above equation holds. That is, $d(O_1, O_2) = \|d(O_1, O') - d(O_2, O')\|$. Hence O' preserves the variance of Φ_i. For any point O'' that is not on the line, we have $d(O_1, O_2) > \|d(O_1, O'') - d(O_2, O'')\|$. For any point O''' on the segment (excluding O_1 and O_2), $d(O_1, O_2) = \|d(O_1, O''') + d(O_2, O''')\|$ implies $d(O_1, O_2) > \|d(O_1, O''') - d(O_2, O''')\|$.

Since the variance along Φ_1 is largest, a point O' on Φ_1 and out of its variance segment preserves the largest variance. Based on the definition of optimal reference point, such a O' is an optimal reference point since it achieves the largest variance.

□

Theorem 1 provides the theoretical reasoning for our one dimensional transformation algorithm by using an optimal reference point, as shown below in Figure 10. Our algorithm mainly consists of two steps: finding an optimal reference point and compute the keys.

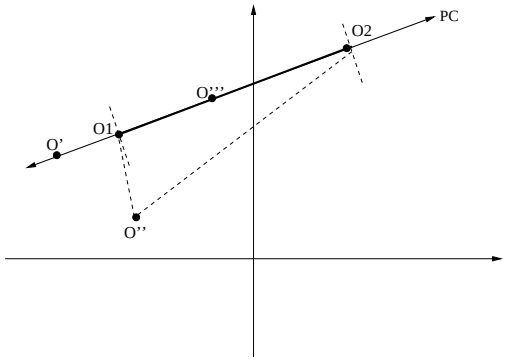


Figure 9: Illustration of variance preservation.

One Dimensional Transformation Algorithm

1. Finding an optimal reference point
   - 1.1 Compute the data center of the dataset
   - 1.2 Compute Φ_1 and its variance segment by PCA
   - 1.3 Compute an optimal reference point O'
2. Generating keys
   - 2.1 for each point O_i, compute its key by $d(O_i, O')$

Figure 10: Mapping by an optimal reference point

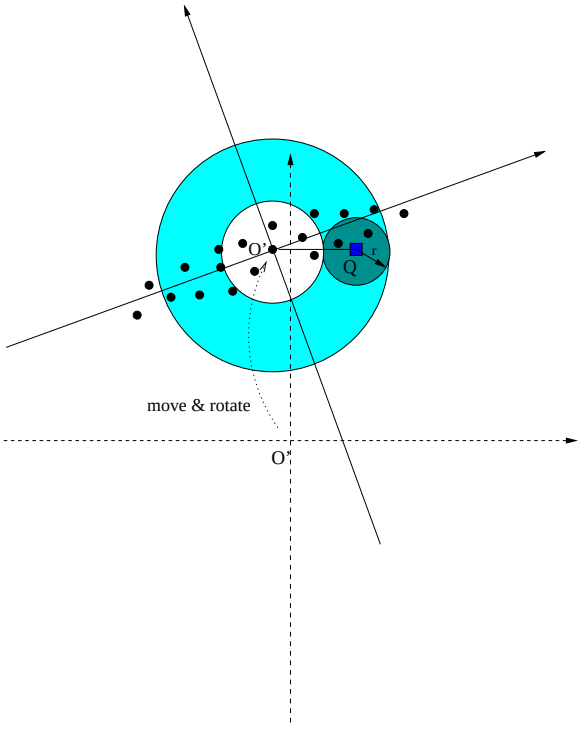


Figure 11: Points accessed by using data center as a reference point

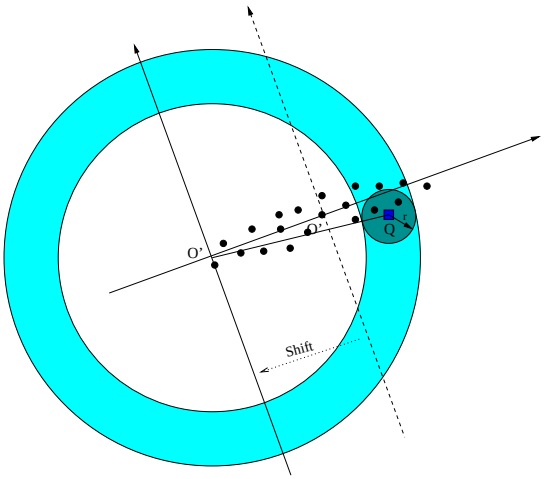


Figure 12: Points accessed by using an optimal reference point

Finding an optimal reference point composes of three substeps. First, the data center is computed. Second, PCA is used to find the first principle component Φ_1 and its variance segment. Third, an optimal point is determined by shifting the data center along the direction of Φ_1 out of Φ_1's variance segment. Geometrically, three substeps can be visualized by changing the original axis system. Computing the data center moves the center of axis system from its origin to the data center, and computing principle components rotates the axis system to make each principle component as one axis in the system (as shown in Figure 11). Locating an optimal reference point is to shift the axis system along the first principle Φ_1 until the center of system is out of Φ_1's variance segment (as shown in Figure 12). After an optimal reference point has been found, the one dimensional transformation by distance function d is then performed to get keys. Finally, a B^+-tree is then built on those keys. Given a query Q and a search radius γ, the portion of points being accessed for a dataset are shown in Figure 8, 11 and 12 when the reference point is space center, data center, and optimal respectively.

In ViTri model, the positions representing N cluster centers are high-dimensional points. Naturally, ViTris' positions are first transformed into one dimensional distance values, followed by being indexed by a B^+-tree. At leaf node level, the volume and density for each ViTri are also stored for similarity computation in query processing.

Dynamic maintenance can be easily handled for indexing. When a new video is added in the dataset, its features are first extracted and summarized into ViTris. The one dimensional key value is then generated for each ViTri by an optimal reference point, followed by the standard insertion operation in B^+-trees by inserting the key into the B^+-tree and ViTri into leaf node.

5.2 KNN Query Processing

In the previous subsection, the ViTris, denoted as ($ViTri_1$, $ViTri_2$,...,$ViTri_N$), are indexed based on the distances of their positions corresponding to an optimal reference point. In this subsection, we discuss how a K-Nearest Neighbors (KNN) query is processed to find the top-K most similar video clips given a query video.

The query video is first summarized into a set of M ViTris based on our clustering algorithm, denoted as ($ViTri_1^Q$, $ViTri_2^Q$,...,$ViTri_M^Q$). In our ViTri similarity measure, the information on positions, radii and densities of two ViTris are taken into account. Particularly, the similarity of two ViTris are zero when there is no intersection (Case 1 in Section 4.2).

For a $ViTris_i$, its radius is at most $\epsilon/2$ based on our clustering algorithm. Hence for a $ViTri_i^Q(O_i^Q, R_i^Q, D_i^Q)$, a $ViTri_i(O_i, R_i, D_i)$ has zero similarity if $d(O_i, O_i^Q) \geq R_i^Q + \epsilon/2$. Based on triangle inequality, a $ViTri_i$ in B^+-tree can be safely pruned away if $\|d(O_i, O') - d(O_i^Q, O')\| \geq R_i^Q + \epsilon/2$. This can be done by performing an efficient range search in B^+-tree, with range of $[d(O_i^Q, O')$-$(R_i^Q + \epsilon/2)$, $d(O_i^Q, O')$+$(R_i^Q + \epsilon/2)]$. That is, for a $ViTri_i^Q$, its search radius γ is $(R_i^Q + \epsilon/2)$. All the ViTris in the range are then evaluated with the similarity with respective to $ViTri_i^Q$.

A naive KNN method is first to compute the similar ViTris for each $ViTris_i^Q$, followed by integrating all the results of ($ViTri_1^Q$, $ViTri_2^Q$,...,$ViTri_M^Q$) to obtain the final KNN results. However, such a method will access B^+-tree M times and many ViTris are repeatedly accessed if the search ranges of any two $ViTri_i^Q$ and $ViTri_j^Q$ overlap. Figure 13 shows an example of complete overlapping between search ranges of $ViTri_i^Q$ and $ViTri_j^Q$. In this case, ViTris accessed by $ViTri_i^Q$ are all re-accessed by $ViTri_j^Q$. To alleviate this problem, we introduce query composition.

Query composition analyzes the search ranges of ViTris in the query video and composes the overlapped ranges into single one to eliminate the duplicate accesses. Assume the search ranges of $ViTri_i^Q$ and $ViTri_j^Q$ overlap and $d(O_i^Q, O')$+$(R_i^Q + \epsilon/2) \geq d(O_j^Q, O')$+$(R_j^Q + \epsilon/2)$, then the composed range is $[\min(d(O_i^Q, O')$-$(R_i^Q$+$\epsilon/2)$,$d(O_j^Q, O')$-$(R_j^Q$+$\epsilon/2))$, $d(O_i^Q, O')$+$(R_i^Q + \epsilon/2)]$. The composed range compares with the rest ranges to perform further composition until no overlap exists. Range searches are then processed for those composed ranges. Notice that query composition reduces the times of accessing B^+-tree and duplicate page accesses on leaf nodes.

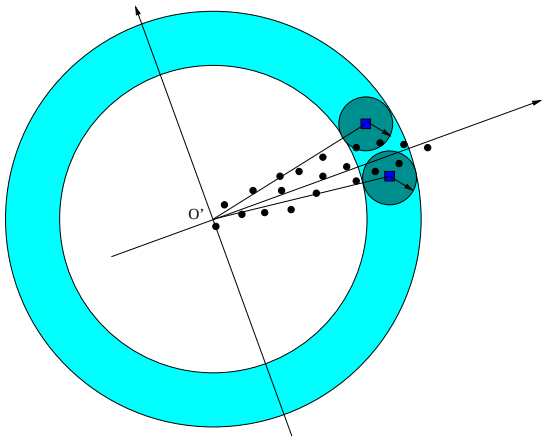


Figure 13: Repeated accesses due to range overlap

6. PERFORMANCE STUDY

In this section, we report the results of an extensive performance study conducted to evaluate the proposed methods on large real video data sets. We also compare with existing video measures and one dimensional transformation methods.

6.1 Experiments Set up

Our dataset consists of about 6,500 video clips which are TV advertisements captured from TV stations. They are recorded by using Virtual Dub at PAL frame rate of 25fps [1]. Each frame is at 192 x 144 pixels resolution and compressed using PLCVideo Mjpegs.

Each frame is represented by a 64-dimensional feature vector in the RGB color space, where two most significant bits are used for each color channel. The value of each dimension is normalized by dividing the total number of pixels. The detailed information of the dataset is listed in Table 2.

Time Length (s)	Number of Video	Number of Frame
30	2934	2,200,482
15	2519	566,772
10	1134	283,486

Table 2: Data statistics

It is impractical to browse very large video datasets to generate the manually-identified ground-truth for a query. In this paper we obtain a query's ground-truth (the top-K best results in KNN search) by comparing the query with database videos at frame level using the similarity measure defined in Section 3.1. Denote the set of ground-truth as rel, and the set of results returned by a summarization method as ret, the precision achieved by the summarization method is defined as:

$$precision = \frac{|rel \cap ret|}{|rel|}$$

All the experiments were performed on a Sun Enterprise E420 (4x450MHz CPU's with 4GB RAM). We used a page size of 4K. All results reported are the average based on 50 queries for 50NN. The distance function d we used is Euclidean function.

Value of ϵ	Number of clusters	Average cluster size
0.2	141,334	22
0.3	69,477	44
0.4	33,285	92
0.5	21,213	168
0.6	9,411	324

Table 3: Summary statistics

6.2 Effectiveness of ViTri

In our method, frame similarity threshold ϵ is the only parameter to be tuned. It affects the number and compactness of clusters for each video sequence. Table 3 shows the statistics on the summary of the video dataset while varying ϵ values. As ϵ increases, the number of cluster decreases significantly due to enlarged cluster radii. When ϵ reaches 0.6, many video clips are summarized into single clusters, and their average number of frame is about 324.

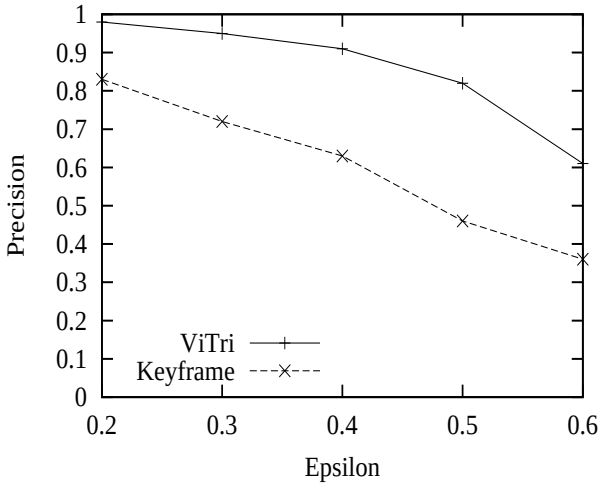


Figure 14: Retrieval precision vs. epsilon

Next, we test the effect of ϵ on the precision of our method. We compare ViTri with existing keyframe method [5] whose video similarity measure is the percentage of similar keyframes. Figure 14 shows the retrieval precision for both methods with a large spectrum of ϵ. We have two main observations. First, the retrieval precisions of both methods drop as ϵ increases. A larger ϵ leads to larger cluster sizes and radii, so as to the cluster volumes. It is expected that clusters with large radii are less compact, thus less accurate in representing the original information. Hence it is reasonable that the precision decreases as ϵ increases. Second, ViTri always outperforms keyframe method by a large margin, and the margin becomes larger as ϵ increases. In addition to the cluster center, ViTri also remembers local information including its volume and density. ViTri similarity is measured by estimating number of similar frames in two clusters, however, existing methods determine the similarity by simply checking whether the distance of the cluster centers is greater than a threshold ϵ. This experiment conforms the superiority of our method over existing ones. To enforce a high retrieval precision, we set ϵ to be 0.3 for the rest of experiments.

We also test the effect of K, the number of return results in KNN search, on the precision with fixed ϵ of 0.3. As we can see from Figure 15, ViTri outperforms keyframe method by a noticeable gap, and the precision is not very sensitive to the change of K. Actually, the precision of ViTri increases slightly as K becomes larger. Based on the definition of precision, single miss in a small K affects the precision more significantly than a large K.

6.3 Efficiency of ViTri Indexing

In the following, we test the performance of our one dimensional indexing method on ViTris.

6.3.1 *Effectiveness of Query Composition*

We first test the effect of different KNN query processing methods: naive method and query composition in B^+-tree. Figure 16 depicts the I/O improvement achieved by query composition over the naive method. This experiment confirms that a large amount of ViTris are repeatedly accessed

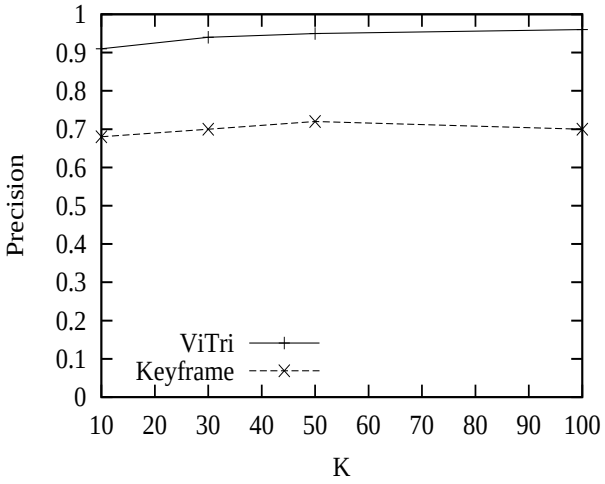


Figure 15: Retrieval precision vs. K

for a query video sequence. Performing KNN Search for ViTris in the query one by one will obviously increase the total number of page accesses (as shown by the naive method). Query composition enables each ViTri in leaf node of B^+-tree to be accessed at most once for a query to reduce the expensive I/O cost as shown in Figure 16.

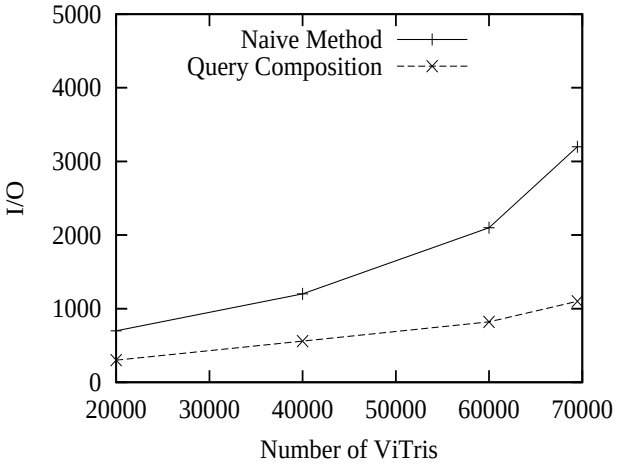


Figure 16: Comparison of query processing methods

6.3.2 A Comparison Study

In this experiment, we compare our method with sequential scan and existing one dimensional transformation method [15] where the data center and space center are chosen as the reference points. We test the effect of number of ViTris and dimensionality of feature spaces.

Figure 17 shows their I/O cost and CPU cost as the number of ViTris increases. It is observed that sequential scan is the worst, followed by data center and space center as the reference point. Transformation by an optimal reference point performs best and outperforms others by 2-5 times. This proves that one dimensional mapping with respective to difference reference points will cause different performances of B^+-tree which is affected by the inter-distances of transformed one dimensional values. An optimal reference point maximizes the variance of inter-distances of one dimensional values and enables B^+-tree to achieve the best performance.

Figure 18 shows their I/O cost and CPU cost as the dimensionality of feature space increases. The similar trends of Figure 17 are observed that both I/O cost and CPU cost increase as the dimensionality increases. Again, transformation by an optimal reference point performs best, followed by data center and space center as the reference point. Interestingly, the I/O cost and CPU cost for one dimensional transformation increase in a faster rate as dimensionality increases. This is reasonable since relatively more information is lost when a higher dimensional vector is transformed into one distance value. Fortunately, an optimal reference point offsets part of negative effect of higher dimensionality and its cost increases in a much slower rate than those of data center and space center.

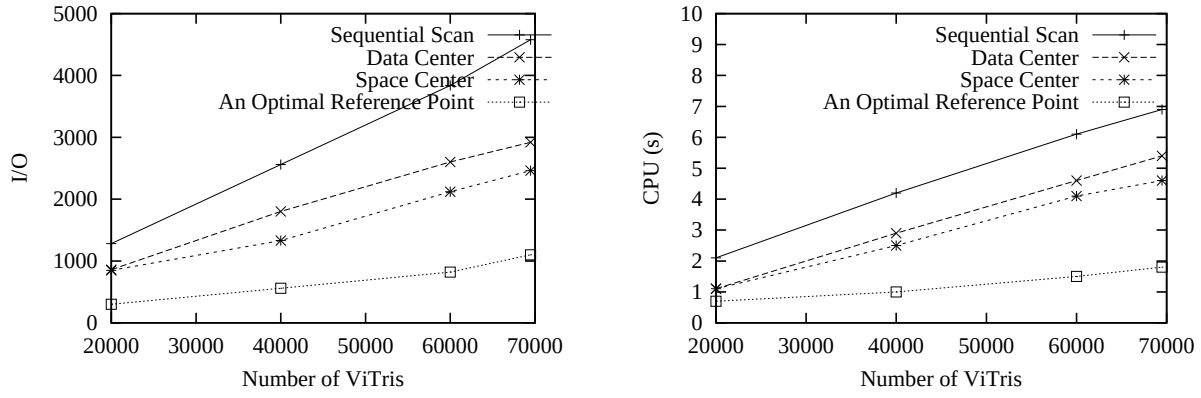
6.3.3 Effect of Dynamic Insertion

In this experiment, we test the effect of dynamic insertion on our indexing method. We initialize the B^+-tree by randomly choosing the first batch of videos which consist of about 20,000 ViTris, then insert other batches of videos which contain about 20,000, 20,000, and 9,477 ViTris respectively. After each insertion of batch videos, we perform KNN search to see how the insertion affects the performance. Figure 19 shows the changing trends of I/O cost and CPU cost for sequential scan and our method as more videos are inserted. Obviously, a larger number of ViTris correspond to a larger B^+-tree which naturally leads to larger I/O and CPU costs. Figure 19 depicts such trends. However, comparing with the linear increasing rate of sequential scan, the increasing rates of our method are much slower.

Compared to the performance of index with one-off construction (i.e., index rebuilt upon each insertion) given in Figure 17, we notice that the performance of our method slightly degrades by dynamic insertion. An optimal reference point is chosen from the first principle component of ViTri dataset. As more ViTris are inserted, the correlation of ViTris may change from original direction. As a result, the original reference point may not be optimal any more. Hence it is reasonable for the performance to degrade slightly. The larger degree of correlation is changed, the farther the reference point away from its optimal position. To preserve the effectiveness of one dimensional indexing, one way is to compute the angle between original first principle component and current first principle component. Once the angle has shifted more than a degree allowed, the index is rebuilt.

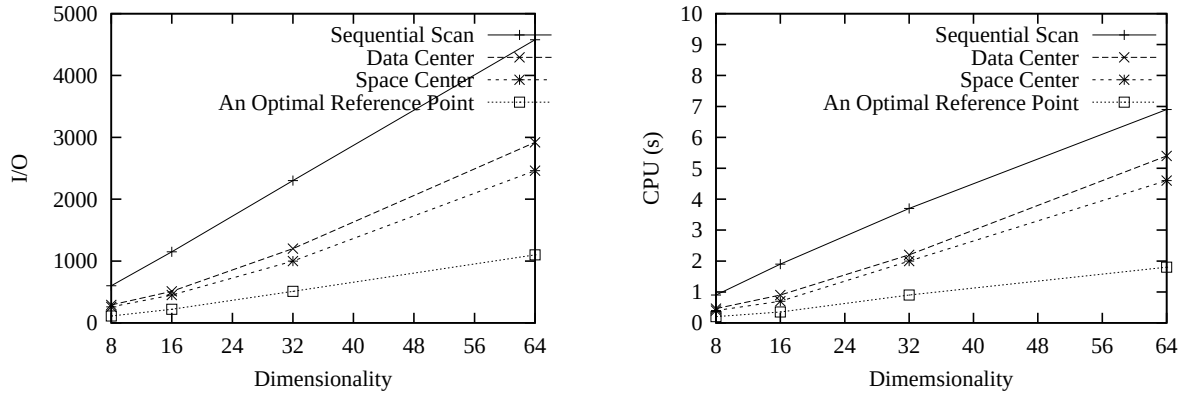
7. CONCLUSION

In this paper, we introduce a novel video summary model called Video Triplet (ViTri). ViTri models a cluster as a tightly bounded hypersphere described by its position, radius, and density. The ViTri similarity is measured by the volume of intersection between two hyperspheres multiplying the minimal density, i.e., the estimated number of similar frames shared by two clusters. The total number of similar frames is then estimated to derive the overall similarity between two video sequences. Hence the time complexity of video similarity measure can be reduced to be linear to the product of number of ViTris from two sequences. To further



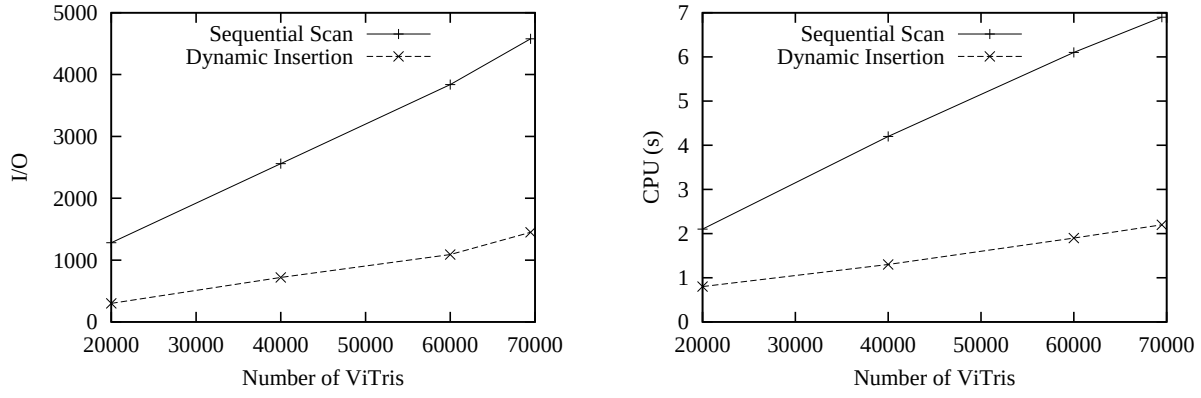
(a) I/O Cost (b) CPU Cost

Figure 17: Effect of Number of ViTris



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

Figure 18: Effect of Dimensionality



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

Figure 19: Effect of dynamic insertion.

speed up the search in ViTris, we introduce a new one dimensional transformation technique which rotates and shifts the original axis system using PCA in such a way that the original variance of inter-distances between points can be maximally retained. By applying this transformation, the ViTris' positions are mapped into one dimensional values which are then indexed by a B^+-tree. Such transformation enables B^+-tree to achieve its optimal performance by quickly filtering a larger portion of non-similar ViTris given a search radius. Our extensive experiments on real large video datasets prove the effectiveness of our proposals that outperform existing methods significantly.

For our future work, we plan to further investigate the effect of data correlation on our one dimensional transformation method. Second, indexing techniques in other categories will be also studied on video sequence database. Third, the sequence alignment and temporal-order will be also considered in video retrieval.

8. ACKNOWLEDGEMENT

We thank Mr Yijun Li for his help in providing video datasets for our experiments.

9. ADDITIONAL AUTHORS

Additional authors: Zi Huang (School of Information Technology and Electrical Engineering,The University of Queensland, email: `huang@itee.uq.edu.au`).

10. REFERENCES

[1] http://www.virtualdub.org/.

[2] S. Berchtold, C. Böhm, and H.-P. Kriegel. The pyramid-technique: Towards breaking the curse of dimensionality. In *SIGMOD*, pages 142–153, 1998.

[3] C. Böhm, S. Berchtold, and D. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, pages 33(3):322–373, 2001.

[4] K. Chakrabarti and S. Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *VLDB*, pages 89–100, 2000.

[5] H. Chang, S. Sull, and S. Lee. Efficient video indexing scheme for content-based retrieval. In *IEEE Transactions on Circuits and Systems for Video Technology*, 1999.

[6] S. Cheung and A. Zakhor. Efficient video similarity measurement with video signature. In *IEEE Transactions on Circuits and Systems for Video Technology*, 2003.

[7] P. Indyk, G. Iyengar, and N. Shivakumar. Finding pirated video sequences on the internet. In *Tech. Rep., Stanford Infolab*, 1999.

[8] G. Iyengar and A. Lippman. Distributional clustering for efficient content-based retrieval of images and video. In *ICIP*, pages 81–84, 2000.

[9] J. Jacquelin. Problem of hyperspace. In *http://maths-express.ifrance.com/maths-express/articles/voir/hyperchevre.htm*, 2004.

[10] C. Jensen, D. Lin, and B.C.Ooi. Query and update efficient b+-tree based indexing of moving objects. In *VLDB*, 2004.

[11] H. Jin, B. Ooi, H. Shen, C. Yu, and A. Zhou. An Adaptive and Efficient Dimensionality Reduction Algorithm for High-Dimensional Indexing. In *ICDE*, pages 87–98, 2003.

[12] S.-W. Kim, C. C. Aggarwal, and P. S. Y. i. Effective nearest neighbor indexing with the euclidean metric. In *CIKM*, pages 9–16, 2001.

[13] M. R. Naphade, R. Wang, and T. S. Huang. Multimodal pattern matching for audio-visual query and retrieval. In *SPIE*, pages 188–195, 2001.

[14] N. Vasconcelos. On the complexity of probabilistic image retrieval. In *ICCV*, 2001.

[15] C. Yu, B. Ooi, K. Tan, and H. V. Jagadish. Indexing the distance: An efficient method to KNN processing. In *VLDB*, pages 166–174, 2001.