Continuous Content-Based Copy Detection over Streaming Videos

Ying Yan [†1], Beng Chin Ooi [‡2], Aoying Zhou [†§3]

[†]*Department of Computer Science and Engineering, Fudan University*
{yingyan, ayzhou}@fudan.edu.cn

[‡]*Department of Computer Science, National University of Singapore*
ooibc@comp.nus.edu.sg

[§]*Institute of Massive Computing, East China Normal University*
ayzhou@sei.ecnu.edu.cn

***Abstract*— Digital videos are increasingly adopted in various multimedia applications where they are usually broadcasted or transmitted as video streams. Continuously monitoring copies on the fast and long streaming videos is gaining attention due to its importance in content and rights management. The problem of video copies detection on video streams is complicated by two issues. First, original videos may be edited, with their frames being reordered, to avoid detection. Second, there are many concurrent video streams and for each stream, there could be many continuous video copy monitoring queries. Efficient data stream algorithms are therefore essential for processing a large number of continuous queries on video streams. In this paper, we first define video sequence similarity that is robust with respect to changes of videos, and a hash-based video sketch for efficient computation of sequence similarity. We then present a novel bit vector signature of the sketch to achieve two optimization objectives: CPU cost and memory requirement. Finally, in order to handle multiple continuous queries simultaneously, we design an index structure for the query sequences. We implemented the system and use real videos for the experimental study. Experimental results confirm the efficiency and effectiveness of our proposed techniques.**[1 2 3]

I. Introduction

With the advent of inexpensive video capturing, authoring and storage devices and with the increasing bandwidth, digital video sharing and broadcasting over the internet has become a reality. Videos are being used increasingly in many applications such as long distance learning, product and service information dissemination and of course, the entertainment. Unfortunately, ease of editing, transmitting and duplicating have also made copyright protection a problem. In typical scenarios, authors of the videos would like to know how their work have been edited and used by others; film producers are eager to know whether their products have been copied and broadcasted in legal manners; and the advertising agencies would like to ensure that their advertisements have been broadcasted on the prime time slot they pay for and without tamper. In this connection, efficient and effective online video copies detection are therefore important. We call these infinite, broadcasting live videos *streaming videos* in this paper. To solve this kind of problem, both digital techniques and data streaming algorithms are needed.

Two obvious techniques that are relevant to copy detection are watermarking and content based image retrieval (CBIR)[1], [2]. Watermarking is by means of hiding some signals into the video, and it relies on the ability to detect from a copy a distinct pattern earlier introduced into the original copy. However, watermark based detection has its problems: (a) Watermark based detection can only be used if the videos are watermarked. In some countries, it is common that the videos are manually captured in the cinema and illegally put into circulations. In such cases, the video content may be the same, but many features have been altered due to recapturing, and the video is not watermarked. (b) The watermark could be attacked and destroyed or distorted during transmission. (c) The third party who has been engaged for video monitoring services may not have the original watermark information. An effective alternative approach to copy detection is to exploit the fact that "video itself is the watermark" [1] and make use of the contents. Indeed, content based video copy detection is more appropriate for a broader range of applications such as advertisement monitoring by the third party. While videos consist of frames and could be segmented based on scenes, direct application of CBIR techniques on video streams for copy detection is however inefficient. First, thousands of hours of videos are broadcasted everyday. It is not practical to store all the videos from different sources. It would be more efficient to process the incoming streams in real time without archiving, and only store the video sequences which are relevant to the queries for further analysis. The techniques in traditional Video Database Management System (VDBMS) cannot be applied directly. Indexing techniques such as [3], [4], [5] are based on the entire sequence comparison in well segmented video databases. Therefore, they are not suitable for copy detection over streaming videos. Second, video editing is common, and many videos may have been temporally reedited before they are republished. For example, some individuals may reorganize

[1]The work of this author was done when she was an intern at National University of Singapore.

[2]The work of this author was in part funded by NUS Research Grant R-252-000-244-112.

[3]The work of this author was partially supported by National Hi-tech R&D Program of China under grant no. 2006AA010111.

the videos along another story line or theme to serve their own purposes. As a result, the temporal features of the shots or even frames of video sequences are changed. Existing subsequence matching techniques such as [1], [6], [7], [8], [9] have not been designed to efficiently detect these temporally reordered video copies. Third, like any other continuous querying systems, there could be many query videos for which the system has to monitor continuously and concurrently over each video stream. Unfortunately, existing work does not consider how to organize the continuous query sequences effectively.

To address the above problem, an efficient Video Data Stream Management System (VDSMS) has to be designed. In this paper, we propose a scheme for fast and robust continuous copy detection over streaming videos. In particular, we have the following contributions.

1) In Section III, we propose a robust compressed domain feature extraction and feature signature encoding scheme for video sequences.
2) In Section IV, we propose an approximate $min\text{-}hash$ based method to construct sketches for the streaming videos. The proposed sketch is not only efficient in terms of video sequence comparison, but also robust for temporally varied video copies.
3) We propose a novel bit signature representation of video sketch, a pruning algorithm and an efficient query index structure in Section V to reduce similarity comparison computation and to provide fast response.
4) In Section VI, we show the experimental results on real videos obtained from Google web site[10]. The results demonstrate the effectiveness and efficiency of our proposed method.

Apart from the main sections mentioned above, we also reviews necessary related work in Section II and conclude the paper in Section VII.

II. Related work

To facilitate meaningful content based video retrieval, it is important to address the problem of video similarity. There are two main approaches. One approach is mainly based on the frame feature of videos. It considers video sequence as a collection of individual frames, and computes the similarity based on the image features. Chang et al.[11] devise a system which retrieves videos by key frame comparison. Cheung et al.[3] develop a randomly seeded frames comparison algorithm. These methods are not efficient because of the large number of image comparisons and high segmentation cost in extracting key frames. Various techniques have also been proposed to improve the efficiency of image feature based comparisons. In [4], the authors propose a novel signature on video sequence based on the percentage of color dominated pixels. Shen et al.[5] propose to compute the similarity based on the volume intersection between two hyper-spheres determined by two video clips to reduce computational cost. The authors of [12], [13], [14] extract fingerprints which are discriminant enough among different images. Indexes can be built over these fingerprints for image or video copy detection. However, the whole video has to be analyzed to obtain feature clusters for indexing purposes. Lu et al.[15] propose ordered VA-file indexing technique and efficient KNN search algorithm to speed up video retrieval and achieve high accuracy.

The other approach is to take the temporal order into consideration, by viewing a video as an ordered sequence of frames, and compare video clips sequence by sequence. Hoi et al.[16] propose a two-step filter-and-refine approach based on nearest feature trajectories. Park et al.[17] propose suffix tree indexing techniques for video or image sequence matching using time warping distance. Diakopoulos et al. [18] propose a method based on sub-shot level segmentation and out of order matches of these segmentation. However, all the methods mentioned above have been designed for full video clip matching and cannot be applied directly to streaming video copy detection problem. Subsequence video matches have higher demands on efficiency because most such applications need real time response. Kim et al.[9] combine ordinal and temporal signatures together for sequence matches to improve the speed. Hampapur et al.[1] propose sequence comparison methods based on motion direction, ordinal intensity and color histogram signatures. In their proposal, query sequence slides frame by frame on data sequence with a fix-sized window. Chiu et al.[6] use warping distance computation on selected critical frames which not only solves the sequence length variations problem identified in [1], but is also more effective than the approach proposed in [9]. Adjeroh et al.[7] turn the video sequence matching into substring matching problem and use edit distance as the similarity measure. Hoad et al.[8] describe novel techniques for extracting features from light centroid and color shifts of video sequence. However, when the temporal pattern is broken by reordering the shots or frames of video sequences, all these subsequence matching techniques have not been shown to be able to make temporal variations effectively. Either, all have not been designed for video stream.

A key step towards efficient video comparison is semantic feature extraction. Videos are encoded into bit streams to ease storage and transmission. There are methods based on pixel domain [4], [5], [11], [16] as well as the compressed domain[19], [20]. Feature extraction on the uncompressed pixel domain can only be applied to applications which do not need real time response. On the contrary, operations on compressed domain avoid costly computation of inverse Discrete Cosine Transform (DCT) and thus achieve higher efficiency.

Processing time and storage are both critical for data stream problems. In this paper, we propose to build sketches over the incoming video streams to achieve high performance. Our sketch is based on a family of hash functions, $min\text{-}wise$ independent, which were introduced in [21] and recently investigated in [22]. The basic idea of $min\text{-}wise$ hashing scheme is that the similarity of two sets of objects is estimated by their independent permutations. Hence, it can be used to measure set similarity in different applicatioins[23], [24]. However, the number of of hash functions could be too large to make it practical, and consequently, approximate $min\text{-}wise$

family with error bound was introduced in [24], [25], [26].

III. Streaming Video Copy Detection

A video stream S is a sequence of frames $S = \{s_1, s_2...s_N\}$. A video subsequence of S from frame s_i to s_j is denoted as $P(i{:}j) = \{s_i...s_j\}$. The length of $P(i{:}j)$ is the number of frames it contains.

Definition 1: Streaming Video Copy Detection. Given a set of query sequences $\psi = \{Q_1, Q_2...Q_m\}$ and a video stream S, if any subsequence $P(i{:}j)$ ($i, j \in [1, N]$ and $i<j$) over S satisfies

$$sim(Q_k, P) \geq \delta \quad k \in [1, m]$$

$P(i{:}j)$ is returned as a copy of query Q_k. δ is the similarity threshold.

A. Frame Fingerprint

In order to facilitate fast video frame comparison, we use signatures to represent the features extracted from the video frames. We propose a two-phase method in deriving the signature: multi-dimensional feature extraction and dimensionality reduction.

Feature Extraction. Image fingerprint extraction techniques such as [12], [13], [14] are robust to geometric transformations and invariant to illumination variations. Naturally, these methods can be imbedded into our system. However, they are more focused on effect rather than on speed. In order to ensure fast online response, we use the following simple and effective feature extraction method on compressed domain. We partially decode incoming video bit streams to Discrete Cosine (DC) sequence and extract the DC coefficients of key (or I) frames. Each key frame is spatially partitioned into D equal sized blocks. The average DC coefficient value of each block is computed. All these D average values in each frame are then normalized to the [0,1] range as below:

$$C_i = \frac{\tilde{C}_i - \tilde{C}_{min}}{\tilde{C}_{max} - \tilde{C}_{min}} \quad (i \in [0, D-1]) \tag{1}$$

where $\tilde{C}_i$, $\tilde{C}_{max}$ and $\tilde{C}_{min}$ are respectively the i_{th}, maximum and minimum values of the D average coefficients. Then, we select d coefficients from D blocks. The advantages of this feature extraction are as follow: (1) Working on the DC domain is significantly faster than on the pixel domain[20], therefore it is ideal for streaming applications for fast responses. (2) Multiple video copies streaming from different sources usually have variations in brightness, color or frame size. However, ordinal relationship of coefficients among blocks in a frame remains stable. After normalization, the coefficient of each block is also relatively stable among different copies.

Dimensionality Reduction. In order to reduce computational cost, and construct sketches for video sequences, we reduce the dimensionality of the normalized feature vector by transforming it to a single dimensional signature. To this end, we partition the feature space into equal sized cells.

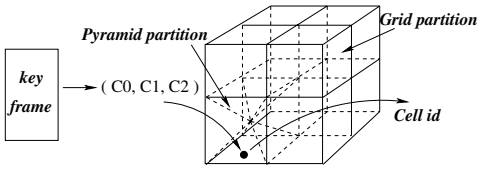


Fig. 1. Feature Space Partition

The signature of each frame is represented by the cell id its feature belongs to. By grid partitioning, the space along each dimension is divided into u equal width segments, we get u^d cells for the d dimensional space. The granularity of the cells affects the number of false positives and false negatives. If the number of cells is too small, many features will be mapped to the same representative cell and this will result in many false positives. If this number is too huge, there are many false negatives because of the small variances of the coefficients among different copies.

Based on the ordinal pattern preserving characteristic of video data, we propose a grid-pyramid based partitioning method. Each dimension is first divided into u slices by grid partition. Then, each grid cell is further divided into $2d$ subcells by pyramid partitioning, as shown in Figure 1. Therefore, the space is partitioned to $2du^d$ cells. Given a frame's feature vector f, the cell is computed from $id = 2dO_g(f)+O_p(f)$, where O_g is grid order or space filling order and O_p is pyramid order. Suppose the center of one cell is $<V_0, V_1...V_{d-1}>$, the pyramid order is as follows[27]:

$$O_p = \begin{cases} j_{max} & : \quad j_{max} < V_{j_{max}} \\ j_{max} + d & : \quad j_{max} \geq V_{j_{max}} \end{cases}$$

$$j_{max} = (j | \forall i, 0 \leq (j, i) < d-1, j \neq i; |V_j - C_j| \geq |V_i - C_i|)$$

In pyramid partition, the cell of a feature point is determined by a single value j_{max}= $arg\{\max\{|V_j - C_j|\}\}$ ($j \in [0, d-1]$).

The rationale behind the use of both grid-based and pyramid-based partitioning is as follows. Pyramid partitioning divides the space only into $2d$ cells. With only $2d$ cell ids, the number of false positives is just too big. Another reason is due to the characteristics of extracted video features. Given the block coefficients of two copies of frames with different compressed settings C=$<C_0, ..., C_{d-1}>$ and C'=$<C'_0, ..., C'_{d-1}>$. The orders of vector C and C' may be similar, and yet are not exactly the same, as in each vector, the ranks of some coefficients are changed because of the slight differences between each pair of these coefficients. Moreover, the fraction of these changed values is usually small[1]. Unless the value j_{max} is changed, variances of other values will not affect the pyramid cell id. Suppose there are k values whose ranks are changed, the probability of changing j_{max} is $1 - \frac{\binom{k}{D-1}}{\binom{k}{D}} = \frac{k}{D}$ which is low due to the robustness of ordinal features. On the contrary, if the grid partitioning alone is used, the slight changes of each dimension's feature value may cause feature points of two copies being mapped to different cells. Therefore, having a pyramid partition in each grid cell will result in less false negatives than just the pure grid-based partitions.

B. Similarity Measure

After assigning a signature to the feature of each frame, a video sequence is mapped to a data sequence $P(i{:}j) = \{s_i ... s_j\}$, where s_i is a 1-dimensional value indicating the cell id of the i_{th} frame. To obtain robustness against temporal variations, we use set similarity as the measure of video sequences' similarity which is,

Definition 2: Video Sequence Similarity. Given two video sequences Q and P, the similarity is defined as:

$$sim(Q, P) = \frac{|Q \cap P|}{|Q \cup P|}$$

The computation of this similarity between P and Q lies in finding the distinct number of common cell ids $|Q \cap P|$. It is not a difficult task when P and Q are two finite video sequences. However, in video streaming setting, membership test for $Q \cap P$ on each arrived frame is not practical. The cost for the brute force search scales linearly to the number of queries as well as the length of each query and cannot meet the requirement of efficiency on high speed streams. Therefore, a sketch technique over streaming video is required.

IV. Sketching Video Sequences

Sketches constructed over streaming videos should facilitate a much more efficient definition 2 computation while not losing much of the information for deriving the actual similarity. In this section, we introduce an effective $min\text{-}hash$ based sketch technique.

Theorem 1: min-wise independent.[22] A family of hash functions $\psi \subseteq S_n$ is min-wise independent if $\forall x \in X$ and set $X \subseteq \{1, 2..., n\}$, π is chosen at random in ψ according to some specified probability distribution, then we get the following:

$$Pr\{\min\{\pi(X)\} = \pi(x)\} = \frac{1}{|X|} \tag{2}$$

The requirement is that there is equal probability that any element in X will become the minimum element of the image of X under π. The distribution on ψ can either be uniform or skewed.

When we randomly select a hash function π from ψ to get a sample $\tau(X) \in X$, $\tau(X) = \pi^{-1}(\min\{\pi(X)\})$, then, for any two non-empty subsets Q and P,

$$Pr\{\tau(Q) = \tau(P)\} = \frac{|Q \cap P|}{|Q \cup P|} \tag{3}$$

However, in order to satisfy equation 2 the exact number of $min\text{-}hash$ function could be very large, resulting in high computational overhead. Consequently, $approximate$ $min\text{-}wise$ hash is introduced [25], [26]. $sim(Q, P)$ is estimated by pre-computing set Q and P's $K\text{-}min\text{-}hash$ sketches $Q' = \{\min\pi_1(Q), ..., \min\pi_K(Q)\}$ and $P' = \{\min\pi_1(P), ..., \min\pi_K(P)\}$. The value of similarity is the percentage of equal $min\text{-}hash$ values between Q' and P'.

A. Streaming Video Subsequences Comparison

Due to different encoding settings such as frame rate, video copies of the same content may not be of the same length. In order to get accurate results, we should compute the similarity of subsequences with query sequences at any possible start position and of any possible length. The subsequences from the video stream, which are potentially to be matched, are called *candidate sequences*. A basic assumption is that, if the length of a query is L, then, the length of candidate sequence is upper bounded by λL. As discussed in[28], the optimal tempo scaling parameter λ is no bigger than 2. Maintaining and testing an overly long subsequence is neither practical nor meaningful, therefore the candidate sequences which are longer than λL in the candidate list C_L, termed *expired sequences*, which will be removed from the list.

We first partition the video stream into small windows with equal size w, which are called *basic windows*. Then, we combine consecutive basic windows into different lengthes of candidate sequences. The candidate sequence which satisfies definition 1 with query sequence Q, is returned as a copy of Q.

The size of each candidate sequence P is the number of basic windows, $|P| \in [1, \lceil \frac{\lambda L}{w} \rceil]$. We denote each basic window as w_i, $P[i, j] = \{w_i \circ w_{i+1} ... \circ w_j\}$, $(i \leq j)$. The sketch of $P[i, j]$ is a vector containing $K\text{-}min\text{-}hash$ values denoted as $sk_{ij} = \{sk_{ij}^1, sk_{ij}^2, ..., sk_{ij}^K\}$. As we shall see, our approximate $min\text{-}hash$ sketch not only makes the similarity comparison easier but also has good properties for bottom up multi-length candidate sequence computation.

Property 1: For any candidate sequence $P[i, j]$, $(i \leq j)$, its approximate $min\text{-}hash$ sketch can be computed by choosing the smallest hash value from the sketches of its subsequences $P[i, x]$ and $P[x, j]$, where $x \in [i, j]$.

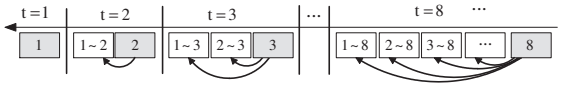
The proof is omitted here due to space limitation.

Based on this property, the sketch of any candidate sequence P can be computed from the combination of basic windows' sketches. We consider two combination orders: *Sequential* and *Geometric Order*. In *Sequential Order*, the i_{th} basic window sketch combines with all the sketches of its previous candidate sequences' in candidate list C_L. Function $\mathbf{Comb}$(A,B) combines sketch B with A and stores the result in A. While in *Geometric Order*, as shown in Figure 2 where sk_i is the sketch of basic window i and C_L is the candidate sequence list. There are only $\lceil \log i \rceil$ combinations when the i_{th} sketch arrives. As illustrated in Figure 3(b), when the 4_{th} basic window arrives, it first combines with candidate sequence 3. The result will combine with candidate sequence 1.

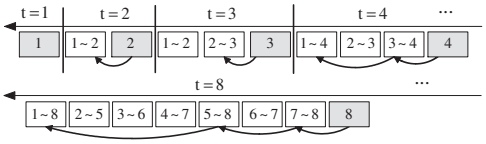
In *Sequential order*, candidate sequences are maintained with size ranging from 1 to $\lceil \frac{\lambda L}{w} \rceil$. It is adopted when the accuracy is of primary importance. *Geometric Order* only stores candidate sequences with size of $log(i)$, $i \in [1, \lceil \frac{\lambda L}{w} \rceil]$. Although it is more efficient than *Sequential order*, it may have more false negatives due to skipping of frames.

Algorithm GeometricOrder
Input: sk_i: Sketch of basic window i
C_L: Candidate sequence list
1. **if** C_L is not empty **then**
2. **for** j=1 to $log(\lceil \frac{\lambda L}{w} \rceil) + 1$ **do**
3. **if** j=1 **then**
4. **Merge**($C_L[\lceil \frac{\lambda L}{w} \rceil - 1], sk_i$);
5. **else Merge**($C_L[\lceil \frac{\lambda L}{w} \rceil - 2^j + 1]$, $C_L[\lceil \frac{\lambda L}{w} \rceil - 2^{j-1} + 1]$);

Fig. 2. Geometric Order



(a) Sequential



(b) Geometric

Fig. 3. Different Combination Orders

B. Cost Analysis

For each basic window, we need to perform three operations: (1) Comparing the basic window sketch with query sketches for similarity of definition 2; (2) Combining the sketch with other candidate sequences in the candidate list and (3) Comparing the combined candidate sequences with query sketches again for similarity. The total processing cost per basic window is:

$$\begin{cases} \alpha C_{comp} + (\alpha C_{comp} + C_{comb})\lceil \frac{\lambda L}{w} \rceil & : \ Sequential \\ \alpha C_{comp} + (\alpha C_{comp} + C_{comb}) \log(\lceil \frac{\lambda L}{w} \rceil) & : \ Geometric \end{cases} \quad (4)$$

C_{comp} is the cost for comparing of two sketches. C_{comb} is the cost of combining a pair of sketches. Independent of the combination order being used, the cost is entirely determined by three factors: (1) Sketch operation cost C_{comp} and C_{comb}. (2) The number of basic windows $\lceil \frac{\lambda L}{w} \rceil$ maintained, where L is the maximum length of queries. (3) The number of queries α we need to compare with. In order to improve the overall efficiency, we must improve the sketch computation and comparison, reduce the amount of candidate sequences maintained, and reduce the number of comparison between candidate sequences and the queries.

V. Optimizing the Operation of Sketch Comparison

A. Bit Vector Signature

When comparing the candidate sequence sketches with the query sketches, we actually just need to count the number of equal hash values instead of identifying what the hash values are. The following example illustrates the case in point. Suppose we choose the r_{th} hash values sk^r_{ix}, sk^r_{xj} and sk^r_q from two candidate sequence sketches sk_{ix}, sk_{xj} and a query sketch sk_q. If sk^r_{ix} is bigger than sk^r_q, we mark a symbol ">" over sk^r_{ix}. If sk^r_{ix} is equal to sk^r_q, we use the symbol "=". We use "<" when sk^r_{ix} is smaller than sk^r_q. Correspondingly, we get the symbols from the comparison between sk^r_{xj} and sk^r_q. Suppose sk_{ij} is the sketch of merging sk_{ix} and sk_{xj}. The symbol between sk^r_{ij} and sk^r_q can be obtained directly from the symbols of sk^r_{ix} and sk^r_{xj}. For example, if sk^r_{ix} ">" sk^r_q, and sk^r_{xj} ">" sk^r_q, then sk^r_{ij} ">" sk^r_q; if sk^r_{ix} "<" sk^r_q, and sk^r_{xj} "=" sk^r_q, then sk^r_{ij} "<" sk^r_q. Therefore, we devise a bit vector signature encoded according to the relationship between the hash values.

Definition 3: Bit Vector Signature. Given the sketch of a candidate sequence sk_{ij} and a query sequence sk_q, the values of the r_{th} hash function ($r \in [1, K]$) are respectively sk^r_{ij} and sk^r_q. $B_{ij.q}$ is a bit vector signature with length $2K$ encoded by the relationship between sk_{ij} and sk_q with the following rules:

$$B_{ij.q}[r-1, r] = \begin{cases} 00 & : \ sk^r_{ij} > sk^r_q \\ 01 & : \ sk^r_{ij} = sk^r_q \\ 11 & : \ sk^r_{ij} < sk^r_q \end{cases}$$

The encoding procedure does not lose any information from sketches because of the followings,

$\min\{>, >\} =$">" $\Longleftrightarrow 00|00 = 00$,
$\min\{>, =\} =$"=" $\Longleftrightarrow 00|01 = 01$,
$\min\{>, <\} =$"<" $\Longleftrightarrow 00|11 = 11$,
$\min\{=, =\} =$"=" $\Longleftrightarrow 01|01 = 01$,
$\min\{=, <\} =$"<" $\Longleftrightarrow 01|11 = 11$,
$\min\{<, <\} =$"<" $\Longleftrightarrow 11|11 = 11$.

When comparing with query q, for any two candidate sequences $P[i, x]$ and $P[x, j]$, the bit signature of their combination $B_{ij.q}$ is the "OR" between their bit signature $B_{ix.q}$ and $B_{xj.q}$. The similarity between the combination $P[i, j]$ and query q is computed according to the following lemma.

Lemma 1: The similarity between a candidate sequence and a query can be computed from their bit vectors. Assume n_0 is the number of 0 on even positions and n_1 is the number of 1 on the odd positions in the bit vector. Then,

$$sim = 1 - \frac{n_0 + n_1}{K}$$

Proof: As in definition 3, n_0 is exactly the number of ">" values, while n_1 is the number of "<" values. Thus, $K - n_0 - n_1$ is the number of equal values. We get the similarity $sim = \frac{K - n_0 - n_1}{K} = 1 - \frac{n_0 + n_1}{K}$. ■

Each sketch $sk_{ij} = \{sk^1_{ij}, sk^2_{ij}, ..., sk^K_{ij}\}$ is represented by a bit vector signature of length $2K$ bits according to a query q. We only need to compare each basic window sketch with query sketches to get the bit signature. Signatures of other candidate sequences can be obtained by bit "OR" of basic windows' signature without accessing the query. The similarity can be computed by counting the number of 0 and 1 according to Lemma 1. The array comparison in equation 4 is transformed to bit operation to reduce the cost of C_{comp} and C_{comb}.

B. Pruning Strategies

Lemma 2: Given a query sequence q with sketch sk_q, and a candidate sequence $P[i, j]$ with sketch sk_{ij}, the bit signature of sk_{ij} corresponding to sk_q is $B_{ij.q}$. If $P[i, j]$ is a copy of

q, the number of 1 on the odd positions of $B_{ij.q}$ must be no bigger than $K(1-\delta)$ with similarity threshold δ.

Proof: From property 1, we know that, sk_{ij} is computed from its candidate sequences sketches, $\forall r \in [1,K], sk^r_{ij} = \min\{sk^r_{ix}, sk^r_{xj}\}$. For a given query sketch sk_q, if $sk^r_{ix} < sk^r_q$, then $sk^r_{ij} < sk^r_q$ for any $j > i$. The reason is that if $sk^r_{xj} \geq sk^r_{ix}$, then $sk^r_{ij} = sk^r_{ix} < sk^r_q$, otherwise $sk^r_{ij} = sk^r_{xj} < sk^r_{ix} < sk^r_q$. The similarity between the candidate sequence and query q is determined by the number of equal $min\text{-}hash$ values N_e. As the size of candidate sequences grows bigger, N_e is varied. However, there are less hash values bigger than query (N_b) and more hash values smaller than query (N_s), which is N_b becomes smaller and N_s is bigger. N_e equals to $K - N_s - N_b$ which is no bigger than $K - N_s$. Thus, we get $N_s \leq K - N_e$. As defined in definition 1, the subsequences which is a video copy must satisfy $N_e \geq K\delta$. Therefore, we conclude N_s must be no bigger than $K(1-\delta)$. According to definition 3, N_s is also the number of 1 in the odd positions of $B_{ij.q}$. ■

A candidate sequence P which does not satisfy Lemma 2 should be removed. As argued in the proof, all the candidate sequences which are in the combination order list of P should also be removed. Because, if N_s in sketch P is big, N_s in its combination sequences will be no less than that in P. Therefore, all the related candidate sequences can be pruned.

The success of bit signature lies in that, it considers the relationship between candidate sequence sketch and query sketch. When there are multiple queries, each candidate sequence does not only maintain the signatures of its related queries but also those related to its consecutive candidate sequences in C_L. Despite the fact that more related query signatures have to be maintained, the cost of using bit signature is still low because: (1) For the matched candidate sequences, their related queries tend to be similar with its consecutive candidate sequences. For those queries that do not match, their signatures will not remain in the list long before being pruned. As such, the signature list is not big. (2) Instead of choosing the maximum query length, L can be selected according to each query. Unqualified signatures related to short length queries will be pruned early to reduce memory consumption. (3) Each signature is $2K$ bits, the total signature size in the candidate list is $2\alpha K$ bits, for α related queries. With our query index structure described in the following section, α is kept very small, and therefore, the memory consumption is small.

C. Indexing Query Sequences

In real applications, there are a large number of video sequences subscribed as continuous queries over streaming videos. Maintaining and comparing candidate sequences with each query is not cost effective. Typically, each candidate sequence is only relevant to one or a small number of queries, efficient indexing of query sequences is therefore essential. In this section, we introduce a simple and yet effective data structure for organizing the sketches of query sequences.

1) The Data Structure: The sketches of the query sequences can be $min\text{-}hashed$ offline. Suppose the number of queries is m and the number of hash functions is K, the number of hash values in all the queries are $m \times K$. For the index construction algorithm where $QS[m][K]$ is Query sketches, we organize them in a two dimensional ***Hash-Query*** array $HQ[K][m]$. Each data element $HQ[i][j]$ is a triple $<value, up, down>$. $value$ is the $min\text{-}hash$ value of the i_{th} hash function of query q. up denotes the position of query q's (i-1)$_{th}$ hash value on (i-1)$_{th}$ row. $down$ represents the position of q's (i+1)$_{th}$ hash value on (i+1)$_{th}$ row. At the entry of each column, there is information about the query id and query length. Data triples on each row are ordered according to $value$.

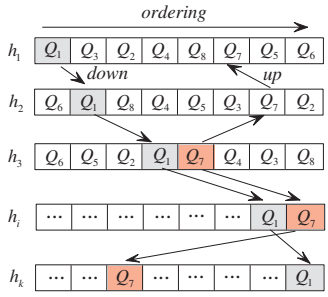


Fig. 4. Index Structure

In this data structure, given the value of the i_{th} hash function, binary search is performed on the i_{th} row. If the matched position is $HQ[i][j]$, up and $down$ search would be performed to find all the other hash values of the same query. When the search reaches the first row, we get the query id and length. Alternatively, given a query id q, from the first row, we can get the position of entry $HQ[0][j]$, then $down$ search is performed to find all the hash values of q. An example index structure is shown in Figure 4, and as can be observed, the hash values of query Q_1 are located at different positions on different rows, but they are connected with the same query id through the value of up and $down$.

Addition of new queries and removal of old queries can be performed online. When a query sequence q is subscribed or unsubscribed, through binary search on each row, proper positions to insert or delete the hash values of q can be determined quickly. up and $down$ should also be updated for those whose positions are changed.

2) Index Probing: Given a basic window sketch sk and query index HQ, the algorithm returns a related query list R_L. Each element in R_L is a triple $<qid, bitsig, lp>$, where qid denotes related query id, and $bitsig$ denotes bit signature according to relationship between query qid and sk as in the definition 3. The search is performed on each hash function $i \in [1,K]$, and lp records the position of its (i-1)$_{th}$ hash value on (i-1)$_{th}$ row.

The query index probing algorithm is outlined in Figure 5 where HQ is the query index structure. For each hash function i, the search procedure consists of the following three major operations. (1) **Bit signature setting** (steps 3-6). For each element ele in R_L, the bit values of $ele.bitsig$ are set according to $HQ[i][HQ[i-1][ele.lp].down].value$ and $sk[i]$. (2) **Relevant queries searching** (steps 12-16). Binary search or equal search should be used to find the positions whose

values are equal to $sk[i]$. These positions correspond to hash values of queries which are not in the current R_L. Suppose a position $HQ[i][j]$ is found, a new element ele_{new} is inserted into R_L. Using up search, the values of $ele_{new}.bitsig$ and $ele_{new}.qid$ are all set. (3) **Pruning** (steps 9-10). Pruning condition of Lemma 2 is enforced in the above two operations in order to remove false positives and negative queries from R_L as early as possible.

For the example shown in Figure 4, when a basic window sketch sk arrives, we find that it has the same hash value with only Q_1 on the first hash function. Then, a bit signature related to Q_1 is inserted into R_L. When the search reaches the third hash function, the hash value of sk is different to Q_1, but it is the same to query Q_7's hash value. Then, a new bit signature related to Q_7 is inserted into R_L. After setting the bit values on the first and second hash functions of query Q_7, two signatures related to both query Q_1 and Q_7 are continued to be tested. Once done, R_L is inserted into C_L. Obviously, the advantage of this data structure lies in that only the relevant queries need to be compared and only their bit vector signatures are maintained, which reduces both the CPU cost and memory consumption.

Algorithm ProbeIndex
Input: sk: Sketch of basic window
HQ: Query index structure
1. **for** each hash function $j \in K$ **do**
2. $Searchflag$ = 0
3. **for** each ele in R_L **do**
4. $t = HQ[i][HQ[i-1][ele.lp].down].value$;
5. $ele.lp = HQ[i-1][ele.lp].down$;
6. Set relation($t, sk[i]$) on $ele.bitsig[2i, 2i+1]$;
7. **if** $t = sk[i]$ **then**
8. $Searchflag = 1$;
9. **if** $ele.bitsig$ does not satisfy Lemma 2 **then**
10. prune;
11. **if** $Searchflag$ = 0 **then**
12. $p_{list} \leftarrow$ **BinarySearch**$(HQ[i], sk[i])$;
13. **else** $p_{list} \leftarrow$ **EqualSearch**$(HQ[i], sk[i])$;
14. **for** each position $HQ[i][j]$ in p_{list} **do**
15. R_L.push(ele_{new});
16. set values on $ele_{new}.bitsig[0...2i]$ and $ele_{new}.qid$;
17. **Return** R_L;

Fig. 5. Index Probing

Due to the space constraint, we shall not outline the algorithm for handling multiple continuous video copy detection queries. Instead, we summarize it below:

1) Construct K-min-hash sketches QS for continuous queries and index structure HQ by **BuildIndex**(QS) offline.
2) Construct K-min-hash sketch sk for every w incoming frames. Through probing query index structure by **ProbeIndex**(sk,HQ), and comparing with its consecutive sequences' related query lists, a related query list R_L is obtained and is included into candidate sequence list C_L.
3) Compute the bit signature of the candidate sequence based on *Sequential Order* or *Geometric Order*. Return it as a detected copy if the the similarity condition is satisfied (cf. Lemma 1). Remove it from C_L if it violates Lemma 2.
4) The process continues till the end of the video stream.

VI. Performance Study

To simulate video streams that contain interesting video subsequences the continuous queries are monitoring, we use 5 films as our base video, and we obtain 200 short videos (NTSC: 352×240, 29.97 fps) from[10], which consist of MTV, advertisements, movie samples and sports short videos and insert them into the base video. The lengths of the inserted videos vary from 30s to 300s. The total length of the "doctored" video is 12 hours long. To study the effectiveness of the proposed video copy detection method, we edit the 200 short videos to simulate two different video streams. The first stream, VS1, is the background film videos with the original 200 short videos randomly inserted. For the second video stream, VS2, we alter 20-50% of the color as well as the brightness, add noises and change the resolutions of the short videos, re-compress them using different frame rate (PAL: 352×288, 25 fps). We partition the edited short videos into segments, reorder these segments without affecting the contents, and randomly insert the short videos into the base video. We also use the 200 short videos as the continuous query videos which are running continuously and concurrently against the two video streams.

For efficiency, the processing time including partial decoding and query processing time which is tested from the arrival of the first frame until the last frame of our base video. Maintenance of query index structure and candidate sequence list C_L consumes memory. Since the size of query index is fixed to $m \times K$ triples when using K hash functions for m queries. We only test the size of C_L in the experiments. Since the size of each bit signature is fixed to $2K$ bits, the memory cost is mainly determined by the number of signatures. We use the average number of bit signatures maintained to evaluate memory consumptions.

To evaluate the effectiveness of the copy detection methods, we measure their *precision* and *recall*. The *precision* is the proportion of video sequences detected by the method which are the copies of queries, while the *recall* is the proportion of the video sequences that are returned by the method. We record the begin $Q_i.begin$ and end $Q_i.end$ positions of query Q_i on the stream. The position where a sequence matches is denoted as $Q_i.p$. If $Q_i.begin+w \leq Q_i.p \leq Q_i.end+w$ holds, this result is correct.

The system is implemented in C++, and all the experiments are conducted on a PC with 2.39GHz CPU and 1GB memory (This is done as a part of PIPA system [29], which has been designed to manage interactive digital media data.) In the experiments, the setting of the parameters are shown in Table I. Without additionally mentioned, the parameters are set as the default values which are chosen based on the experimental results of each subsection.

TABLE I

PARAMETER SETTINGS

Parameter	Range	Default vlaue
Number of hash function K	100-3000	800
Dimensionality d	3-7	5
Partition u	2-7	4
Number of Query m	10-200	200
Simiarity threshold δ	0.5-0.9	0.7
Size of basic window w	5s-20s	5s

A. Effects of Space Partitions

In this group of experiments, we want to evaluate the effectiveness of different number of space partition u on different dimension d. We partition each frame into 3×3 blocks and extract $d \in [3, 7]$. The original 200 short videos(A) in VS1 and 200 edited videos(B) in VS2 are used as data sets. We use $A[i]$ and $B[i]$ to represent the i_{th} short video sequence in A and B. As introduced above, videos $A[i]$ and $B[i]$ have the same contents but different encoding settings. We use each short video $A[i]$ as query to search the videos in B using membership test method instead of $min\text{-}hash$. Given dimension d, we examine the *precision* and *recall* with varied u from 2 to 7. As shown in Table II, when u and p are small, the volume of each grid cell is big, frame features from different video sequences may fall in the same cell. Therefore, *precision* is low but *recall* is high. When u and p are both big, the volume of each grid cell becomes smaller. Thus, *recall* becomes low while *precision* increases. For each d, we should choose u to make both *recall* and *precision* high. Without loss of generality, in the following experiments, we choose u = 4 and d = 5 which has relatively better result but is not the optimal value pair.

TABLE II

PRECISION (p) AND RECALL(r) WITH DIFFERENT u AND d

u	2		3		4		5		6		7	
d	p	r	p	r	p	r	p	r	p	r	p	r
3	0.5	0.995	0.705	0.975	0.815	0.935	0.9	0.915	0.93	0.91	0.935	0.89
4	0.675	0.975	0.895	0.935	0.96	0.905	0.975	0.88	0.98	0.87	0.99	0.86
5	0.81	0.935	0.965	0.905	**0.975**	**0.9**	0.985	0.865	0.99	0.835	0.995	0.82
6	0.875	0.895	0.98	0.89	0.995	0.885	0.995	0.85	0.99	0.825	0.995	0.785
7	0.94	0.825	0.99	0.8	0.995	0.765	0.995	0.735	0.995	0.7	0.995	0.66

B. Effects of Number of Hash Functions

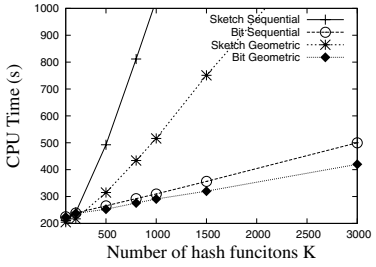


Fig. 6. Effects of K on CPU Time

In this experiment, we want to study the effects of the number of hash functions on the performance of bit signature (Bit) and sketch representations (Sketch). For both representations, we maintain the query structure, and use VS1 as the video stream.

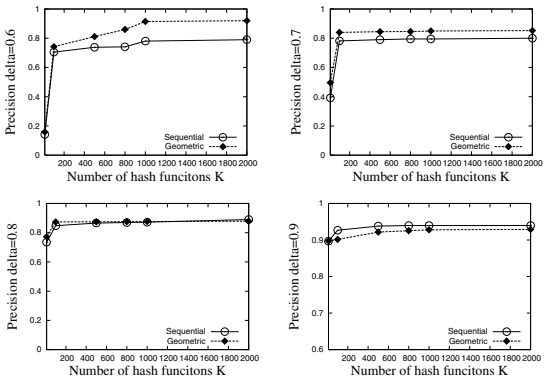


Fig. 7. Effects of K on Precision

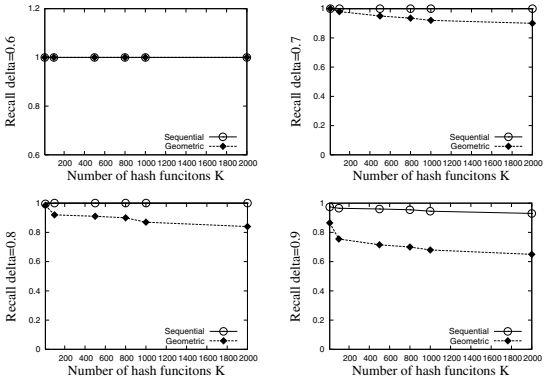


Fig. 8. Effects of K on Recall

We vary the number of hash functions K from 100 to 3000 and fix other parameters to the default values in Table I. The CPU time of the two methods are summarized in Figure 6. When K becomes larger, the cost of sketch operations increases dramatically compared to fast bit operations. Its performance therefore degrades at a faster rate as K increases. For Sketch method, *Geometric Order* is more efficient than *Sequential Order*. But for Bit method, *Geometric Order* saves only a little time. The reason is that, for *Geometric Order*, though the number of combination is small, pruning condition is also tested less frequently which makes C_L larger. Because of the high cost of sketch operation, the reduced cost in combination is substantial, and as a result *Geometric Order* is much faster in Sketch method. However, when bit signature is adopted, the processing time is more related to efficient pruning rather to bit operations.

Since bit signature does not lose any information we need over sketch, we test *precision* and *recall* with varying K from 10 to 2000 only on Bit method when different similarity threshold δ is selected. As shown in Figure 7, as K increases, the *precision* improves. After K reaches 1000, the *precision* improves much more slowly and stays high. As shown in Figure 8, *recall* keeps still or decreases with the increasing of K. The reason is that, when K is small, a few values of features will be selected as samples, video sequences with different contents have more probability to be matched together. Therefore, *precision* is low while *recall* is high. When K is big, these false matched copies cannot match again,

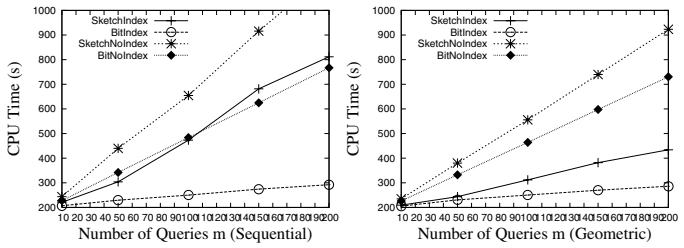
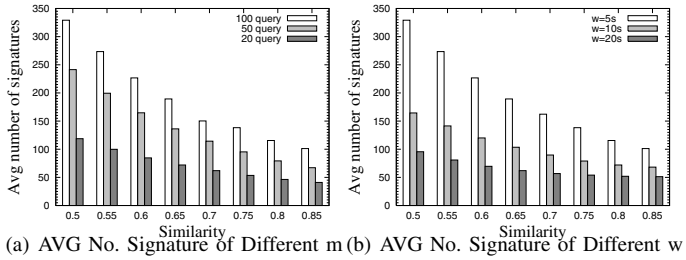


Fig. 9. Effects of Index Structure



(a) AVG No. Signature of Different m (b) AVG No. Signature of Different w

Fig. 10. Memory Consumption

which increases *precision* but lowers *recall*.

Another interesting result we obtain from Figure 7 and 8 is that, *Geometric Order* has higher *precision* than *Sequential Order* when δ is lower, and has lower *recall* when δ becomes big. As we have discussed in Section IV, *Geometric Order* has less matched sequences than *Sequential Order*, when δ is small, there are many false matched copies, *Geometric Order* returns less these copies which enable higher *precision*. When δ is big, the false copies of *Sequential Order* is smaller, but the incomplete results of *Geometric Order* reduce its *precision* and *recall*.

C. Effects of Query Index

In this series of experiments, we study the performance of query index structure on four methods: Sketch with (SketchIndex) and without (SketchNoIndex) index; Bit with (BitIndex) and without (BitNoIndex) index on both *Sequential* and *Geometric* Orders. We fix other parameters to the default values in Table I with varying m. The results are reported in Figure 9. We observe that as m increases, the CPU time for the methods without indexes increase rapidly. On the contrary, it does not affect those with indexes greatly. In *Geometric Order*, SketchIndex is even faster than BitNoIndex after the number of queries is bigger than 100, which also shows the effects of our index structure.

D. Memory Efficiency

In the following experiments, we examine the memory consumption with varying similarity threshold δ, the number of query and size of basic window on VS1 and VS2.

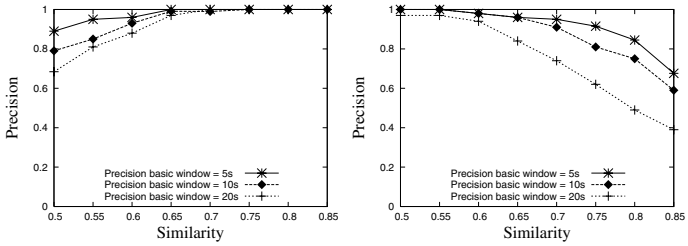


Fig. 11. Precision, Recall of Different w

The method we use is BitIndex with *Sequential order*. We fix other parameters to default values in Table I with varying similarity threshold δ from 0.5 to 0.9. Because the results on VS1 and VS2 are quite similar, we only show the figures for VS2. The average number of bit signatures n is reported in Figure 10(a). As δ increases, the number of related query becomes smaller, n therefore decreases. When δ=0.7, n=150, the average memory consumption is only 30K bytes for 100 queries. In Figure 10(b), we also change the basic window size from 5s to 20s. When the size of basic window is large, each candidate sequence tends to have more distinct frames. The number of false positive and negative queries decreases, and memory consumption is therefore reduced. However, as the size of the basic window becomes bigger, the length of candidate sequence increases rapidly, thus, the value of *precision* and *recall* also decrease as shown in Figure 11. The results confirm that the proposed method is indeed memory efficient.

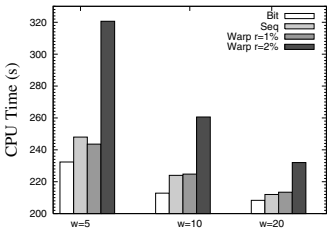


Fig. 12. CPU Time Comparison

E. Comparison with Existing Approaches

In this group of experiments, we compare our proposal against two existing approaches using VS2 query video stream in order to show the ability of our technique in detecting video sequences whose temporal order has been tampered with. We compare with two recent video subsequence matching approaches. The first one is[1] (Seq), in which the similarity is defined as the average distance between each pair of frames in two video sequences. The other is by using time warping distance[6] (Warp), which is more robust to local temporal variations. In time warping distance measure, there

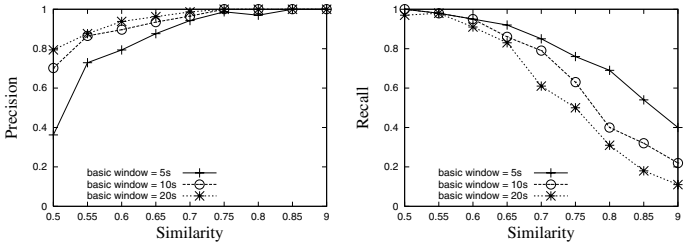


Fig. 13. Similarity Performance of Bit

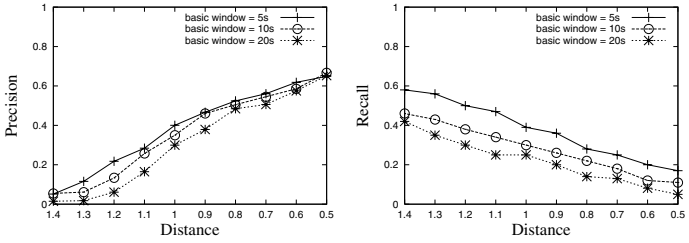


Fig. 14. Similarity Performance of Seq

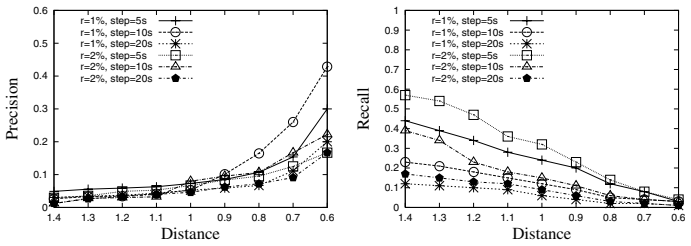


Fig. 15. Similarity Performance of Warp

is a parameter r determining the width of warping path. As r increases, this measure can tolerate more local differences, while the CPU time also increases rapidly. For both of the methods, a query length sized window is sliding through the video stream, the sliding gap (number of jumped frames)is also known as *basic window*. To provide a fair comparison, we also use our compressed domain feature extraction method and make the best implementation of the matching procedure.

We vary the size of the basic window of all the methods and vary the warping width parameter r in Warp method. As shown in Figure 12, our method is the fastest under the setting of different window sizes. The results in Figure 13 show that our method (Bit) achieves high accuracy. For the Seq method, the results from Figure 14 show that the *precisions* increase with decreased distance threshold. However, before the *precisions* reach 50%, the *recalls* of Seq fall below 30%. The performance of this method is affected by the fact that its similarity measure strongly relies on the temporal order of video sequences. Even for warping distance which is meant to be able to tolerate local temporal variances, the precisions and recalls are affected when temporal variations exist and the results are illustrated in Figure 15. In summary, the experiments show that while our proposed method is robust for detecting temporally reedited video copies, the two existing methods, Seq[1] and Warp[6], do not perform as well.

VII. Conclusions

Following the advancement in video editing tools, videos can be copied, edited and reused with ease. Video copy detection and advertisement monitoring over the video streams have therefore emerged to be a lucrative industry. One of the main challenges is to detect video subsequences that have been edited or reordered and subsequently embedded in the video streams In this paper, we present a novel video copy detection method based on the use of $min\text{-}wise$ hash functions for video sketching, a novel bit signature, pruning strategies and query index structure to optimize CPU cost and memory requirement. We conducted extensive experimental studies using real videos, and the results confirm the effectiveness and efficiency of the proposal.

References

[1] A. Hampapur, K. Hyun, and R. M. Bolle, "Comparison of sequence matching techniques for video copy detection," in *Proc. SPIE Vol. 4676, p. 194-201*, 2001.

[2] A. Joly, O. Buisson, and C. Frélicot, "Statistical similarity search applied to content-based video copy detection." in *ICDE Workshops*, 2005, p. 1285.

[3] S.-C. S. Cheung and A. Zakhor, "Efficient video similarity measurement and search," in *ICIP*, 2000, pp. 85–89. [Online]. Available: citeseer.ist.psu.edu/cheung00efficient.html

[4] Y. Li, J. S. Jin, and X. Zhou, "Matching commercial clips from tv streams using a unique, robust and compact signature." in *DICTA*, 2005, p. 39.

[5] H. Shen, B. C. Ooi, and X. Zhou, "Towards effective indexing for very large video sequence database," in *SIGMOD*, 2005, pp. 730–741.

[6] H.-A. W. C.-S. C. AChih-Yi Chiu, Cheng-Hung Li and L.-F. Chien, "A time warping based approach for video copy detection," in *ICPR*, 2006.

[7] D. A. Adjeroh, M. C. Lee, and I. King, "A distance measure for video sequences," *CVIU*, vol. 75, no. 1–2, pp. 25–45, 1999. [Online]. Available: citeseer.ist.psu.edu/adjeroh99distance.html

[8] T. C. Hoad and J. Zobel, "Fast video matching with signature alignment," in *MIR '03*, 2003, pp. 262–269.

[9] C. Kim and B. Vasudev, "Spatiotemporal sequence matching for efficient video copy detection." *IEEE Trans. Circuits Syst. Video Techn.*, vol. 15, no. 1, pp. 127–132, 2005.

[10] "http://video.google.com/."

[11] S.-F. Chang, W. Chen, H. J. Meng, H. Sundaram, and D. Zhong, "Videoq: An automated content based video search system using visual cues," in *ACM Multimedia*, 1997, pp. 313–324. [Online]. Available: citeseer.ist.psu.edu/article/chang97videoq.html

[12] J. Oostveen, T. Kalker, and J. Haitsma, "Feature extraction and a database strategy for video fingerprinting." in *VISUAL*, 2002, pp. 117–128.

[13] C. Schmid and R. Mohr, "Local grayvalue invariants for image retrieval." *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 5, pp. 530–535, 1997.

[14] H. V. Zhao, M. Wu, Z. J. Wang, and K. J. R. Liu, "Forensic analysis of nonlinear collusion attacks for multimedia fingerprinting." *IEEE Transactions on Image Processing*, vol. 14, no. 5, pp. 646–661, 2005.

[15] H. Lu, B. C. Ooi, H. T. Shen, and X. Xue, "Hierarchical indexing structure for efficient similarity search in video retrieval," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 11, pp. 1544–1559, 2006.

[16] C.-H. Hoi, W. Wang, and M. R. Lyu, "A novel scheme for video similarity detection." in *CIVR*, 2003, pp. 373–382.

[17] S. Park and W. W. Chu, "Similarity-based subsequence search in image sequence databases." *Int. J. Image Graphics*, vol. 3, no. 1, pp. 31–54, 2003.

[18] N. Diakopoulos and S. Volmer, "Temporally tolerant video matching," in *Proc. of the ACM SIGIR Workshop on Multimedia Information Retrieval*, 2003. [Online]. Available: citeseer.ist.psu.edu/diakopoulos03temporally.html

[19] V. Kobla, D. S. Doermann, K.-I. Lin, and C. Faloutsos, "Compressed-domain video indexing techniques using dct and motion vector information in mpeg video." in *SPIE*, 1997, pp. 200–211.

[20] V. Mezaris, I. Kompatsiaris, N. V. Boulgouris, and M. G. Strintzis, "Real-time compressed-domain spatiotemporal segmentation and ontologies for video indexing and retrieval." *IEEE Trans. Circuits Syst. Video Techn.*, vol. 14, no. 5, pp. 606–621, 2004.

[21] K. Mulmuley, "Randomized geometric algorithms and pseudo-random generators (extended abstract)," in *FOCS*, 1992, pp. 90–100.

[22] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations (extended abstract)." in *STOC*, 1998, pp. 327–336.

[23] M. Charikar, "Similarity estimation techniques from rounding algorithms." in *STOC*, 2002, pp. 380–388.

[24] M. Datar and S. Muthukrishnan, "Estimating rarity and similarity over data stream windows." in *ESA*, 2002, pp. 323–334.

[25] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang, "Finding interesting associations without support pruning," *Knowledge and Data Engineering*, vol. 13, no. 1, pp. 64–78, 2001. [Online]. Available: citeseer.ist.psu.edu/article/cohen99finding.html

[26] Indyk, "A small approximately min-wise independent family of hash functions," in *SODA*, 1999. [Online]. Available: citeseer.ist.psu.edu/indyk99small.html

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

[28] A. W. chee Fu, E. Keogh, L. Y. H. Lau, and C. A. Ratanamahatana, "Scaling and time warping in time series querying," in *VLDB*, 2005, pp. 649–660.

[29] "http://www.comp.nus.edu.sg/ pipa/."