# EFFICIENT VIDEO RETRIEVAL BY LOCALITY SENSITIVE HASHING

*Shiyan Hu*

Department of Computer and Information Science
Polytechnic University
Brooklyn, NY 11201
shu@cis.poly.edu

## ABSTRACT

In this paper, a new scheme for fast video retrieval is proposed. In the scheme, a video is represented by a set of feature vectors which are computed using the robust Alpha-trimmed average color histogram. To efficiently retrieve videos, the locality sensitive hashing technique, which involves a uniform distance shrinking projection, is applied. Such a technique does not suffer from the notorious "curse of dimensionality" problem in handling high-dimensional data point set and guarantees that geometrically close vectors are hashed to the same bucket with high probability. In addition, unlike the conventional techniques, the involved similarity measure incorporates temporal order of video sequences. The experimental results demonstrate that the proposed scheme outperforms the conventional approaches in accuracy and efficiency.

## 1. INTRODUCTION

Retrieving videos with visually similar content is a very active research topic. Numerous algorithms such as [1, 2, 3] have been proposed for efficient video retrieval in recent years. Typically, prevailing algorithms first map video frames to a feature vector space, and then visual similarity is measured there. To compute the precise value of similarity is usually computationally expensive, which is especially the case for query in large databases of high dimension. On the other hand, the precise value of similarity is not required since feature vectors do not entirely capture the characteristic of human visual system [1]. Therefore, it is unnecessary to maintain full fidelity to the feature vector representations, and approximation can be applied to reduce computation cost [1]. Clearly, a major challenge in content-based video retrieval is how to efficiently identify videos similar to a given query. While the naive approach of sequential search is typically too inefficient to handle large databases, the well-known spatial access methods (e.g., SR-tree) which have been extensively studied by the database community are useful in the context. Most spatial access methods, however, suffer from the notorious "curse of dimensionality" in handling high-dimensional data point set [4].

Recently, a new technique called *locality sensitive hashing* has been proposed in [5] and improved in [6] for the purpose of devising efficient algorithm for high-dimensional nearest neighbor search. The technique theoretically enables us to achieve worst-case $O(dn^{1/(1+\epsilon)})$ time for approximate nearest neighbor query over an $n$-point $d$- dimensional database. This yields an approximate nearest neighbor algorithm running in *sublinear* time for any $\epsilon > 0$, excluding the preprocessing time [6]. [6] also experimentally shows that the method gives significant improvement in running time over other methods such as SR-tree for search in high-dimensional space.

Based on the locality sensitive hashing technique, we propose a new method for efficient content-based video retrieval in this paper. In the method, a video is represented by a set of feature vectors which are then *approximated* for high efficiency through a uniform distance shrinking projection (which is a dimension reduction technique) involved in locality sensitive hashing. Compared to other approaches, the new method uses alpha-trimmed average (ATA) histograms which outperform the conventional histograms in robustness to the adverse effects of brightness/color variations, occlusion, and edit effects on the color representation. In addition, the proposed technique is robust against temporal re-ordering of the video. In particular, the new method runs considerably faster than conventional methods. Finally, it is worth mentioning that [11] explores LSH in image retrieval, however, neither ATA histogram nor temporal information in video are considered.

The rest of the paper is organized as follows: Section 2 gives a high level description of the proposed video retrieval algorithm. Section 3 describes the locality sensitive hashing technique. Section 4 presents the experimental results.

## 2. FAST VIDEO RETRIEVAL ALGORITHM

### 2.1. Alpha-Trimmed Average Color Histogram

Assume that each video sequence is represented by a set of high-dimensional feature vector. Color histogram is one of the most commonly used image features in content-based retrieval systems. [1] uses 178-bin color histogram on the HSV color space to represent the quadrant of each frame in a video and the quantization of the color space used in the histogram is similar to [3]. However, applying simple histograms like this is not robust (see [7, 2]) to the adverse effects of brightness/color variations, occlusion, and edit effects on the color representation. For this reason, we adopt a more robust feature called *"Alpha-trimmed average"* (ATA) histogram [7]. [7] investigates such a histogram and shows that it outperforms commonly used *key-frame* histogram. In contrast to key-frame histogram, ATA histogram represents the cumulative color information of all frames with a *group of frames* (GoF). To this effect, a straightforward way is to accumulate all pixel color values from all frames within a GoF into a single histogram. The normalization version of the resulting histogram is simply the *average (mean) histogram*. A potential problem for average histogram is the sensitivity of the mean operator to outliers, as discussed in [7]. A common solution for this problem is resorting to *median* histogram rather than average histogram. In [7], a family of ATA histograms is defined, which is generated using the trimmed mean operator [8]. An ATA histogram is computed by sorting the array of frame histogram values for each bin in ascending order and averaging central members of the ordered array. Each bin $j$ in the ATA histogram is computed by [7]

$$\alpha TrimHist(j,\alpha) = \frac{1}{M - 2\cdot\lfloor \alpha M \rfloor} \sum_{m=\lfloor \alpha M \rfloor + 1}^{M-\lfloor \alpha M \rfloor} \tilde{h_j}(m) \quad (1)$$

where $M$ is the number of frames in the GoF, and $\tilde{h_j}(m)$ is the sorted array of frame histogram values for the $j$th bin. The trimming parameter $\alpha$ ($0 \leq \alpha \leq 0.5$) controls data points excluded from the average computation, and the trimmed mean operator reduces the adverse effect of aberrant data points through discarding an equal number of samples at either end of the sorted series [7]. Note that when $\alpha = 0$ (resp. $\alpha = 0.5$), the resulting histogram is equivalent to the average histogram (resp. the median histogram). It is demonstrated in [7] that compared to using any single frame histogram from GoF, ATA histogram eliminates the effects of undesirable luminance, chrominance variations within the GoF, and the true color content of the GoF is accentuated by modifying $\alpha$.

In our system, we group every $L_T$ (to be discussed soon) frames into a GoF and to incorporate spatial information, each frame and thus the GoF is partitioned into four equal-sized parts. An ATA histogram is computed for each of the four parts of GoF. The resulting four histograms are concatenated to form a feature vector of the GoF.

### 2.2. Algorithm At A High Level

A video is represented by a set of feature vectors, each of which corresponds to a GoF. Basically, two video sequences are similar if most of their feature vectors are geometrically close. To efficiently measure the similarity, we must be able to efficiently determine the nearest neighbors for a query point. For this, standard spatial query techniques may be applied. However, such techniques are too computationally expensive and they do not perform well in high dimensional metric space [4].

To efficiently determine the nearest neighbors of a high-dimensional query point, we resort to the *locality sensitive hashing* (LSH) technique, which has an interesting feature that geometrically close vectors are hashed to the same bucket with high probability. Suppose all videos in the database are hashed into an LSH $H$. Given a query video consisting of $w \cdot T_L$ frames, we are to find the most $s$ similar videos in the database. Assume that $w$ feature vectors $f_1, f_2, \ldots, f_w$ of the query video are hashed into the buckets $q_1, q_2, \ldots, q_w$, respectively. Since geometrically close items (i.e., the similar frames) are expected to lie in the same bucket, we only inspect vectors in buckets $q_1, q_2, \ldots, q_w$. A video is similar to the query if they have many similar feature vectors, so the most $s$ similar videos can be simply computed as those with $s$ largest numbers of similar feature vectors. To improve accuracy, we compute the distances between $f_i$ and feature vectors in bucket $q_i$ for each $i$ (note that if a video does not have a feature vector in $q_i$, the distance is $\infty$ for this bucket), and compute the best $s$ videos by summarizing information from all buckets. That is, the similarity of a video to the query is determined by the sum of the inverse of the distances corresponding to this video in each $q_i$ ($i = 1, \ldots, w$). To further reduce the effect of aberrant frames, the worst and the best few distances are removed from measuring the similarity.

### 2.3. Incorporate Temporal Information

A noteworthy issue is that temporal information is incorporated into the proposed method since temporal order of video frames is important for some applications. To this effect, a straightforward and common solution is to return more candidate videos by modifying the system as a coarse level retrieval followed by the fine level retrieval for the resulting videos. The fine level retrieval can use the techniques such as [2] which takes the temporal-order information into account. The modified system should work well since the frames in similar video should be similar even

if they are perturbed in temporal order. However, a better strategy is to incorporate temporal information directly into the system (i.e., everything is carried out in "coarse level"). From the analysis in [9, 10], the ordinal feature has superior performance to the motion and color feature for the purpose of video copy detection. Basically, such a feature combines both of the color and *spatial* properties of the video. Following [10], our algorithm would ideally go as follows. A frame $f_i$ is partitioned into four blocks and its ordinal feature is computed by the ATA histogram $h_{i,j}$ ($j = 1, 2, 3, 4$) for each block $j$ (note that for simplicity, we denote $\alpha TrimHist(j)$ by $h_j$). Given two videos $a$ and $b$ and their ordinal features $fa_i, fb_i$ for each frame $i$, the distance between two videos at any time point (i.e., frame in this case) $t$ is naturally defined by

$$Dist(t) = \frac{1}{L_T} \sum_{i=t-\lfloor L_T/2 \rfloor}^{t+L_T-\lfloor L_T/2 \rfloor-1} |fa_i - fb_i| \qquad (2)$$

where $L_T$ denotes the comparison period. Loosely speaking, the above definition says that two videos are similar at a time point if a few frames close to this time point are similar.

Recall that a video is partitioned such that every $L_T$ frames form a GoF. Each GoF is then mapped to a high-dimensional feature vector by its four ATA histograms. The above definition troubles us when comparing two feature vectors. Recall that we compute only one histogram $\widehat{h_j(t)}$ for all $j$th quadrant frames in the GoF, which can be loosely regarded as the average of histograms for each individual frame upon normalization, i.e., $\widehat{h_j(t)} \approx \frac{1}{L_T} \sum_{i=t-L_T/2}^{t+L_T/2} h_j(i)$, $j = 1, 2, 3, 4$. However, Eqn. (2) requires computing the distance between each individual frames of two videos. Though such distance is bounded *below* by the distance between histograms $\widehat{ha_j(t)}$ and $\widehat{hb_j(t)}$, the latter, which is used in our system, well approximates the former since consecutive frames in a video are usually similar to each other.

## 3. LOCALITY SENSITIVE HASHING

*Locality Sensitive Hashing* (LSH) is a relatively new scheme for approximate similarity search based on hashing. The basic idea is to hash the points such that "the probability of collision is much higher for points that are close to each other than for those that are far apart" [5]. It was originally introduced in [5] and then improved in [6] for the purpose of devising efficient algorithm for nearest neighbor search. In particular, it enables us to achieve worst-case $O(dn^{1/(1+\epsilon)})$ time for approximate nearest neighbor query over an $n$-point $d$-dimensional database. This yields an approximate nearest neighbor algorithm running in sublinear time for any $\epsilon > 0$, excluding the preprocessing time [6]. In [6], they show that the method gives significant improvement in running time over other methods for search in high-dimensional space based on hierarchical tree decomposition such as SR-tree. For completeness, some details of computing LSH in [6] are included as follows.

We begin with several definitions in [6]. In the *K-Nearest Neighbor search* (KNN), we wish to return the $K$ points in the database that are closest to the query point. The approximate version $\epsilon$-KNN of the KNN problem is that we wish to return $K$ points $p_1, \ldots, p_K$ such that the distance of $p_i$ to the query point $q$ is at most $1 + \epsilon$ times the distance from the $i$th nearest point to $q$. Recall that the distance is defined by $l_1$ norm in our case. Denote by $C$ the maximum possible histogram value.

We first embed our $d$-dimensional point set $P = \{p\}$ into the Hamming cube $H^{d'}$ with $d' = Cd$, by transforming each point $p = (x_1, \ldots, x_d)$ into a binary vector $v(p) = Unary_C(x_1), \ldots, Unary_C(x_d)$ where $Unary_C(x)$ denotes the unary representation of $x$, i.e., a sequence of $x$ ones followed by $C - x$ zeroes. It is easy to see that for any pair of points $p, q$, $d_{L_1}(p, q) = d_H(v(p), v(q))$ [6]. That is, the embedding preserves the distances between the points. The hash functions are defined as follows. For an integer $l$ to be specified later, choose $l$ subsets $I_1, \ldots, I_l$ of $\{1, \ldots, d'\}$. Let $p|I_j$ denote the projection of vector $p$ on the coordinate set $I_j$. Denote $g_j(p) = p|I_j$. For this, we can think of $g_j(p)$ as a dimension-reduction mapping of $p$.

For the preprocessing, we store each $p \in P$ in the bucket $g_j(p)$, for $j = 1, \ldots, l$. Since the total number of buckets (for each $j$) may be large, we compress the buckets by resorting to standard hashing. LSH uses two levels of hashing and we maintain a two-level hash table $\mathrm{T}_j$ for each $j$. Let $B$ denote the maximum bucket size of the latter hash table. If a bucket size is larger than $B$, a new bucket of size $B$ is allocated and linked to and from the old one. To process a query $q$, we search all indices $g_1(q), \ldots, g_l(q)$ until we encounter $K$ points or use all $l$ indices (note that we differ from [6] in this, which leads to good results in our case). It remains to specify the choice of the subsets $I_j$. For each $j \in \{1, \ldots, l\}$, the set $I_j$ consists of $k$ elements randomly sampled from $\{1, \ldots, d'\}$. We are to choose optimal values of parameters such that the probability that a point $p$ close to $q$ will fall into the same bucket as $q$ is maximized, and the probability that a point $p'$ far away from $q$ will fall into the same bucket is minimized. Refer to [6] for further details.

## 4. EXPERIMENTAL RESULTS

In the experiments, we investigate the efficiency of the proposed method versus some existing techniques. In our video database, we collect about 500 video clips with length ranging from 1 minutes to 20 minutes. Some of them are downloaded from the Web, and some of them are sampled from

**Table 1**. Comparison of algorithms in accuracy and query time. In LSH-based technique, $T_L = 4$.

| Algorithm | Accuracy | Avg. query time |
|---|---|---|
| LSH-based ($l = 2$) | 0.85 | 0.20 |
| LSH-based ($l = 3$) | 0.88 | 0.31 |
| LSH-based ($l = 4$) | 0.90 | 0.38 |
| FastMap-based | 0.89 | 0.43 |
| PCA-based | 0.85 | 0.96 |

**Table 2**. Comparison of effects by different $T_L$ in LSH-based technique, $l = 4$.

| $T_L$ | Accuracy | Avg. query time |
|---|---|---|
| 1 | 0.83 | 0.23 |
| 2 | 0.85 | 0.27 |
| 4 | 0.90 | 0.38 |
| 8 | 0.90 | 0.46 |
| 16 | 0.90 | 0.70 |

the same sources with different coding formats, resolutions, and slight color modifications. In our experiment, ATA color histogram (with $\alpha = 0.2$) is extracted as the low level feature for similarity measure and all histogram computations are carried out in the YCbCr color space. The luminance component is coarsely quantized to four uniformly spaced bins to reduce sensitivity to variations in lightness. Each chrominance channel is quantized to eight bins, yielding a total of 256 bins for each histogram. Recall that each frame is partitioned into four parts, so the dimension of each feature vector is 1024.

For comparison, we carry out the experiments using three algorithms, namely, the LSH-based algorithm with different $l$ (recall that $l$ is the number of hash tables each vector is hashed to), FastMap [12]-based algorithm and PCA-based algorithm. Note that for LSH-based algorithm, $l_1$ distance is used, but for FastMap and PCA-based algorithm, $l_2$ distance is used. As two well-known dimension reduction techniques, FastMap and PCA are applied directly to 1024-dimensional feature vectors. For fairness, after dimension reduction, the resulting lower-dimensional vectors are processed into an LSH to support efficient spatial search. Other SAM data structures are also applicable. To compare the performance by using LSH and other SAMs is interesting, however, this issue lies outside the scope of this paper. We refer the interested readers to [6] for such comparisons.

In the experiment, 100 queries are randomly sampled from the database for testing. The average efficiency and accuracy for each technique is summarized in Table 1. Note that in Table 1, $T_L$ is set to 4, which is experimentally determined as the best $T_L$. For the proposed LSH-based method, we find that the best $l$ is 4.

A final remark is on determining the best $T_L$ in the proposed technique. We compare the effect by using different $T_L$. Refer to Table 2 where we experimentally determine that the best $T_L$ for the LSH-based method is 4.

## 5. REFERENCES


[1] S.-C. Cheung and A. Zakhor, "Efficient video similarity measurement with video signature," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 1, pp. 59–74, 2003.

[2] C. Hoi, W. Wang, and M. Lyu, "A novel scheme for video similarity detection," *Proceedings of International Conference on Image and Video Retrieval (CIVR)*, pp. 373–382, 2003.

[3] J. R. Smith, "Integrated spatial and feature image systems: Retrieval, compression and analysis." *Ph.D. thesis Graduate School of Arts and Sciences, Columbia University, February, 1997*.

[4] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," *VLDB*, pp. 194–205, 1998.

[5] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," *STOC*, pp. 604–613, 1998.

[6] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," *VLDB*, pp. 518–529, 1999.

[7] A. Ferman, A. Tekalp, and R. Mehrotra, "Robust color histogram descriptors for video segment retrieval and identification," *IEEE Transactions on Image Processing*, vol. 11, no. 5, pp. 497– 508, 2002.

[8] J. Bednar and T. Watt, "Alpha-trimmed means and their relationship to median filters," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-32, no. 1, pp. 145–153, 1984.

[9] A. Hampapur, R. Bolle, and K.-H. Hyun, "Comparison of sequence matching techniques for video copy detection," *Proc of SPIE: Storage and Retrieval for Media Databases*, 2001.

[10] A. Hampapur and R. Bolle, "Comparison of distance measures for video copy detection," *International Conference on Multimedia and Expo (ICME)*, 2001.

[11] P. Indyk and N. Thaper, "Fast color image retrieval via embeddings," *Workshop on Statistical and Computational Theories of Vision (at ICCV)*, 2003.

[12] C. Faloutsos and K.-I. Lin, "FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pp. 163–174, 1995.