HIERARCHICAL, NON-UNIFORM LOCALITY SENSITIVE HASHING AND ITS APPLICATION TO VIDEO IDENTIFICATION

Zixiang Yang
Inst. for Infocomm Research, Singapore
zixiang@i2r.a-star.edu.sg

Wei Tsang Ooi
School of Computing
National Univ. of Singapore
weitsang@nus.edu.sg

Qibin Sun
Inst. for Infocomm Research, Singapore
qibin@i2r.a-star.edu.sg

ABSTRACT

Searching for similar video clips in large video database, or video identification, requires finding nearest neighbor in high-dimensional feature space. Locality sensitive hashing, or LSH, is a well-known indexing method that allows us to efficiently find approximate nearest neighbor in such space. In this paper, we address two weaknesses of LSH when applied to the video identification problem. We propose two enhancements to LSH, and show that our enhancements improve the performance of LSH significantly in terms of efficiency and accuracy.

1. INTRODUCTION

The problem of content-based video identification concerns identifying the source of a given short query video clip in a large video database based on content similarity [1, 2, 3, 4]. Video identification has many applications, including news report tracking on different channels, video copyright management on the Internet, detection and statistical analysis of broadcasted commercials, video database management, etc. Two key steps in building a video database for video identification are (i) feature extractions from each of the video in the database, (ii) indexing of the feature vectors to allow efficient search of similar video.

This paper focuses on building efficient indexing structure for video identification. We first show that *approximate ε-nearest neighbor search*, or ε-NNS [5], is an appropriate method for identifying similar frames in two video clips. Here, an ε-nearest neighbor (ε-NN) is a neighbor of the query point that is within a factor of (1+ε) of the distance to the true nearest neighbor. We then present a well-known ε-NSS indexing structure called locality sensitive hashing, or LSH. We argue that LSH is good for indexing uniformly distributed high-dimensional points, but is not suitable for video identification where data points maybe clustered. We propose two enhancements to LSH. By building a hierarchical hash table, we adapt the number of dimensions hashed based on the density of points. We also choose the hashed dimensions carefully such that the points are more evenly hashed, thus reducing the number of comparisons needed when searching for the nearest neighbor. Experimental results verify that our enhancements improve LSH significantly in both accuracy and efficiency.

The rest of this paper is organized as follows. Section 2 defines the problem of video identification and shows that ε-NNS can be applied to solve the video identification problem with low probability of errors. We describe locality sensitive hashing and our improvements in Section 3. Our experimental results are presented in Section 4 and the future work is in Section 5.

2. VIDEO IDENTIFICATION

2.1. Measuring Video Similarity

In [4], Sen-Ching Chung gave a video similarity definition between two short video clips. We adapt it to judge if a short query video clip X is contained in a long database video clip Y. The definition is based on the percentage of visually similar frames between query video X and database video Y. We first find the total number of frames from X that have visually similar frames in Y, and then compute the ratio of this number to the number of frames in X.

Let $X = \{x_1, x_2, ..., x_m\}$ and $Y = \{y_1, y_2, ..., y_n\}$ be two video sequences, $x_i \in X$ and $y_i \in Y$ are frames, and $|X| \ll |Y|$ where $|X|, |Y|$ denote the length of videos X and Y respectively. Let $d(x, y)$ be the dissimilarity between frame x and frame y, and δ_{th} be a small threshold that measures if two frames are visually similar or not. Then, the video similarity between X and Y, denoted $S(X, Y, \delta_{th})$, can be defined as follows:

$$S(X, Y, \delta_{th}) = \frac{\sum_{x \in X} T(x)}{|X|} \quad (1)$$

where x is any frame in video X and

$$T(x) = \begin{cases} 1 & \text{if } \min_{y \in Y} d(x, y) \leq \delta_{th} \\ 0 & \text{if } \min_{y \in Y} d(x, y) > \delta_{th} \end{cases} \quad (2)$$

From the above definition, the value for $S(X, Y, \delta_{th})$ falls between 0 and 1, and $S(X, Y, \delta_{th}) = 1$ implies that X

0-7803-8603-5/04/$20.00

is contained in Y. To locate video X in Y, we denote the set of all the visually similar frames of X in Y as:

$$Y_{sim} = \{\bar{y}_j | \bar{y}_j = \arg\min_{y \in Y} d(x_j, y), j = 1, 2, .., |X|\} \quad (3)$$

Then, we compare all the equal length video segments in Y around every frame $\bar{y} \in Y_{sim}$ with X. The video segment with the smallest difference is the best match.

2.2. Using ε-NNS for Video Identification

Based on the above video similarity definition, a key to compute $S(X, Y, \delta_{th})$ and Y_{sim} is finding the closest frame $y \in Y$ for every frame $x \in X$, which is equivalent to nearest neighbor search problem in the feature vector space. Since the number of dimensions involved in the feature space is usually very large for video applications, finding the nearest neighbor efficiently is a difficult problem. We argue that finding an approximate nearest neighbor is good enough. Even though using approximate nearest neighbor will introduce some errors in video identification, it will not affect the results much.

Let $d_\varepsilon(x, Y)$ be the distance of the approximate nearest neighbor from x to points in Y. By the definition of ε-NNS [5], we have

$$\min_{y \in Y} d(x, y) \leq d_\varepsilon(x, Y) \leq (1 + \varepsilon) \min_{y \in Y} d(x, y) \quad (4)$$

Suppose we use d_ε instead of true nearest neighbor to compute $S(X, Y, \delta_{th})$, $T(x)$ in equation (2) becomes:

$$T_\varepsilon(x) = \begin{cases} 1 & \text{if } d_\varepsilon(x, Y) \leq \delta_{th} \\ 0 & \text{if } d_\varepsilon(x, Y) > \delta_{th} \end{cases} \quad (5)$$

From equations (4) and (5), we can see that $T_\varepsilon(x) = T(x)$ when $d_\varepsilon(x, Y) \leq \delta_{th}$ or $d_\varepsilon(x, Y) > (1 + \varepsilon)\delta_{th}$.

Hence, by not using true nearest neighbor, we may introduce error only when $\delta_{th} < d_\varepsilon(x, Y) \leq (1 + \varepsilon)\delta_{th}$. Since both ε and δ_{th} are small, the probability of $d_\varepsilon(x, Y) \in (\delta_{th}, (1 + \epsilon)\delta_{th}]$ is small as well.

Another reason why we can use ε-NNS is this. Suppose a query frame $x \in X$ has the true closest frame $y_t \in Y$. Because of the continuity of videos, the frames around y_t are similar to y_t, and therefore similar to the query frame x. The ε-NNS approach is likely to choose these frames as the nearest neighbor. In other words, if ε-NNS find the wrong closest frame $y_w \in Y$, y_w is likely to be a frame around y_t. Therefore, using ε-NNS will not greatly affect the locating position of video X in Y.

3. LOCALITY SENSITIVE HASHING

3.1. Description of Locality Sensitive Hashing

In the previous section, we show that ε-NNS can be used to effectively search for similar videos with low probability of errors. We now describe a well-known method for ε-NNS, called locality sensitive hashing, or LSH [5].

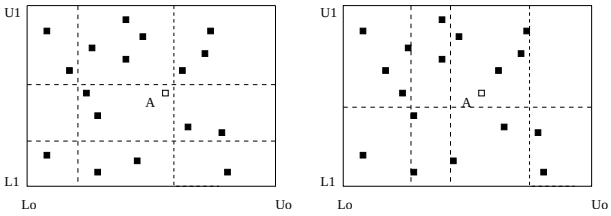

Fig. 1: Locality Sensitive Hashing

The idea behind LSH is rather simple. It randomly partitions a high-dimensional space into high-dimensional cubes. Each cube is a hash bucket[1]. A point is likely to be located in the same bucket as its nearest neighbor. Given a query point, we determine which bucket the point is located in, and perform linear search within a bucket to find the nearest neighbor. The hash function is therefore a mapping from the high-dimensional point to the bitstring representing the bucket the point is in. It is possible that we may miss the nearest neighbor if a point has been hashed to a different bucket than its nearest neighbor (e.g. point A in Figure 1, left). To reduce the likelihood of this, LSH maintains multiple hash tables, hashing a point multiple times using different hash functions. The probability that a point and its nearest neighbor are hashed into different buckets for all these hash functions can be reduced by reducing the number of buckets and increasing the number of hash tables.

We can now describe LSH more formally. Let d be the dimension of the vector space, and $[L_i, U_i]$ be the range of possible values for dimension i. Each hash table in LSH is parameterized by k, the number of hashed dimensions, $D = \langle D_0, D_1, .., D_{k-1} \rangle$, the hashed dimensions, and $T = \langle t_0, t_1, .., t_{k-1} \rangle$, a *threshold vector*. D_i is chosen uniformly at random from $[0, d - 1]$ while t_i is randomly chosen from $[L_{D_i}, U_{D_i}]$.

Given a point $P = \langle p_0, p_1, ..., p_{d-1} \rangle$, we hash it into a k-bit bitstring $b = \langle b_0, b_1, ..., b_{k-1} \rangle$ representing the bucket, where b_i is 1 if $p_{D_i} > t_{D_i}$ and is 0 otherwise.

LSH builds N such hash tables, each with different D and T. The values of N and k can be tuned to change the error bound ε.

Figure 1 illustrates LSH in 2-dimensional space with $k = 4$ and $N = 2$.

3.2. Improvements to Locality Sensitive Hashing

The major factor that determines the efficiency of LSH is the size of the bucket the query points hashed to. Since

[1] In practice, we may hash the bitstring representing the cube using traditional hash functions, resulting in multiple cubes in a bucket.

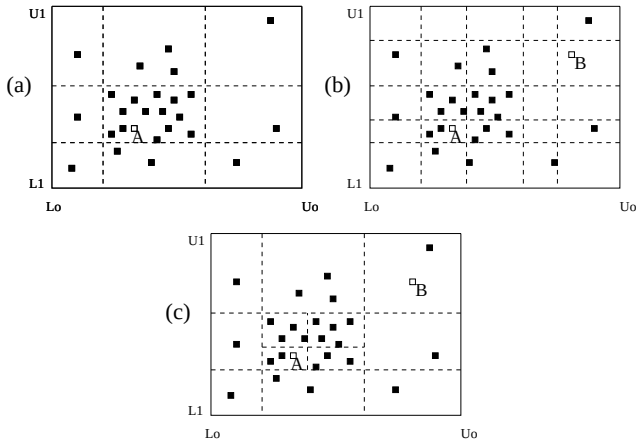

Fig. 2: Hierarchical Partitioning in Locality Sensitive Hashing

for each query point, we need to search through all points in the same bucket to find the nearest neighbor. We would like the points to be evenly distributed among the buckets. However, LSH does not always give such distribution. In this subsection, we illustrate two such problems with LSH and propose two improvements to it.

Hierarchical LSH Currently, LSH partitions the space without considering the distribution of points. In most cases, the real dataset is not uniformly distributed [6]. For example, in Figure 2(a), we see that the number of points in the middle bucket is large. Searching for nearest neighbor of point A will involve many comparisons, thus reducing the efficiency of LSH. One way to solve this problem is to increase k, the number of hashed dimensions. The resulting partitions are shown in Figure 2(b). While this reduces the number of points in each bucket, it reduces the accuracy as well since some query points in sparse area such as point B will miss the true nearest neighbor.

Our solution to this problem is illustrated in Figure 2(c). When the number of points hashed to a bucket exceeds a threshold, we repartition the bucket, and rehash all points in this bucket. This scheme establishes a hierarchy of hash tables in dense area. It reduces the size of the candidate set for linear search while keeping the error low.

LSH with non-uniform partition Another problem of LSH is that the space is partitioned randomly using uniform distribution. This works well when the values of each dimension are evenly distributed. In real dataset, points may be denser in one dimension compared to another. For example, in the case of video identification, some features may be more sensitive than others in differentiating frames. Figure 3(a) illustrates the problem.

To solve the second problem, we should choose the partition dimensions D_i according to the distribution of values in that dimension. Densely distributed dimensions should

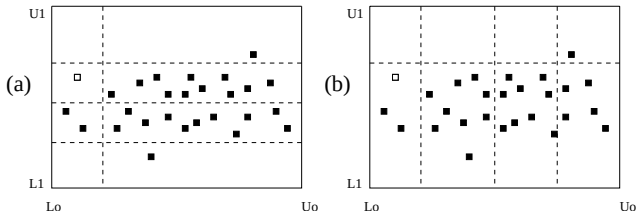

Fig. 3: Non-uniform Selection of Partitioned Dimensions in Locality Sensitive Hashing

be chosen with lower probability while dimensions with uniformly distributed values should be chosen with higher probability. In the example shown in Figure 3, it is better to partition the horizontal dimension compared to the vertical dimension.

We can prove that to reduce the probability that we miss the true nearest neighbor, we should partition the dimension whose values' distribution is closer to uniform distribution with higher probability. However, maintaining the distribution of every dimension is too costly. We choose to use the normalized distribution variance σ^2 as a criterion. Normally, for nearly unimodally distributed dataset, if the distribution of one dimension is close to uniform distribution, its variance is large. So we set the probability of selecting dimension j in proportion to its distribution variance σ^2, i.e.

$$P\{\text{choose } j\} = \frac{\sigma_j^2}{\sum_{i=0}^{d-1} \sigma_i^2}$$

where the denominator is the sum of the variance for all d dimensions.

We call our improved LSH, "hierarchical, non-uniform locality sensitive hashing", or HNLSH.

4. PERFORMANCE EVALUATION

4.1. Performance of HNLSH

To evaluate the performance of our improvements, we implemented the original LSH plus the improvements, and use them to index feature vectors for video identification purposes. We use 6 MPEG news video clips, about 3 hours in length in total. We decode the I frames and use 52-bin normalized color histograms on the Hue-Saturation-Value (HSV) color space to represent each frame (same as [7]). We get 15177 feature vectors with 52 dimensions each. We apply LSH and HNLSH to search the nearest neighbor on this feature vectors dataset. Each indexing structure consists of 4 hash tables ($N = 4$). For each experiment, we vary the number of hashed dimensions (k) from 10 to 30, and issue 200 queries. The performance measured is the average over the 200 queries.

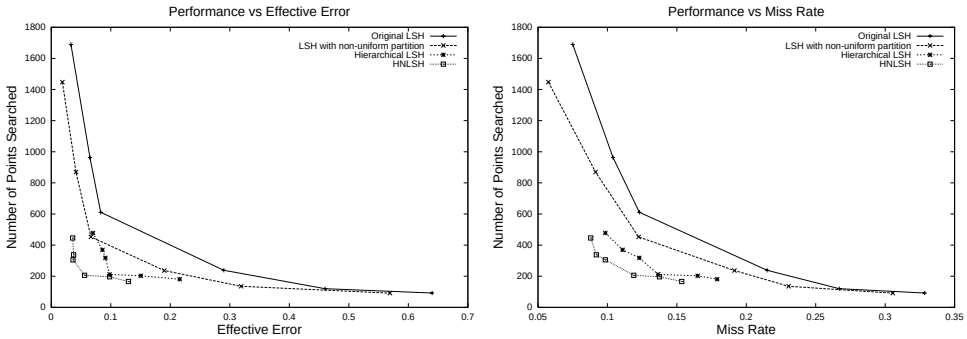

Fig. 4: Experimental Results

We use the size of the bucket which the query point is hashed to as a measurement for efficiency. The error is measured in two ways. One is the *effective error* [5] for ε-NNS,

$$E = \frac{1}{|Q|} \sum_{q \in Q} \frac{d_\varepsilon - d_{min}}{d_{min}}$$

where d_ε denotes the distance from a query point q to a point found by ε-NNS, d_{min} is the distance from q to the closest point, and Q is the set of all query points.

The other error measurement is the rate of which we miss the nearest neighbor, or *miss rate*,

$$R = \frac{1}{|Q|} \sum_{q \in Q} U(q),$$

where $U(q)$ is 1 if $d_\varepsilon > d_{min}$ and is 0 otherwise.

We compare the cost of linear search and error metrics for 4 different implementations of LSH: the original, LSH with hierarchical partitioning, LSH with non-uniform selection of partitioned dimensions, and LSH with both improvements combined (HNLSH). Figure 4 shows our results.

Compared with LSH, HNLSH is significantly better in terms of both performance and accuracy. With HNLSH, we can get the effective error of 0.056 by only 206 linear searches, about 1.3% of the whole dataset size.

4.2. Performance of Video Identification

We use 12 news video clips, with total length of 6 hours to build our video database. 10 query clips of about 5 seconds each are extracted from the database and transcoded to different spatial and temporal resolutions. There are 18 matches from the database. Using HNLSH with $N = 4$ and $k = 10$, we successfully locate 17 correct matches. Each query takes 8.34 milliseconds on the Pentium4, 1.7GHz machine with 384MB memory. This preliminary result is very encouraging and we are currently building a larger video database to further evaluate the performance.

5. FUTURE WORK

We are currently comparing the performance of HNLSH with VA$^+$-file [8] and VQ-index [9], two recently proposed methods for indexing multimedia database. We are also studying how changes to the points distributions, due to database updates, can affect the quality of the hash tables.

Finally, we would like to thank Dr.Wong Lim Soon for his many helpful suggestions and comments on this research.

6. REFERENCES

[1] M. Naphade, M. Yeung, and B. Yeo, "A novel scheme for fast and efficient video sequence matching using compact signatures," in *SPIE, Storage and Retrieval for Media Databases*, Jan 2000.

[2] A. Hampapur and R. Bolle, "Feature based indexing for media tracking," in *Proceedings of ICME*, Aug 2000.

[3] J. Oostveen, T. Kalker, and J. Haitsma, "Feature extraction and a database strategy for video fingerprinting," *LNCS 2314*, pp. 117–128, 2002.

[4] S. C. Cheung and A. Zakhor, "Efficient video similarity measurement with video signature," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 59–74, Jan 2003.

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

[6] N. Katayama and S. Satoh, "The SR-tree: An index structure for high-dimensional nearest neighbor queries," in *Proceedings of SIGMOD*, May 1997, pp. 369–380.

[7] J. R. Smith and S. F. Chang, "Tools and techniques for color image retrieval," in *SPIE, Storage and Retrieval for Image and Video Databases*, 1996, pp. 426–437.

[8] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. El Abbadi, "Vector approximation based indexing for non-uniform high dimensional data sets," in *Proceedings of CIKM*, Nov 2000, pp. 202–209.

[9] E. Tuncel, H. Ferhatosmanoglu, and K. Rose, "VQ-index: An index structure for similarity searching in multimedia databases," in *Proceedings of ACM Multimedia*, Dec 2002, pp. 543–552.