Near-Duplicate Subsequence Matching for Video Streams

Chih-Yi Chiu[†], Yi-Cheng Jhuang[†], Guei-Wun Han[†], and Li-Wei Kang[‡*]
[†] Department of Computer Science and Information Engineering, National Chiayi University, Chiayi City, Taiwan.
[‡] Graduate School of Engineering Science and Technology-Doctoral Program, and Department of Computer Science and Information Engineering, National Yunlin University of Science and Technology, Yunlin County, Taiwan
cychiu@mail.ncyu.edu.tw; s1010428@mail.ncyu.edu.tw; s1010425@mail.ncyu.edu.tw; *lwkang@yuntech.edu.tw

***Abstract*—In this paper, we study the efficiency problem of near-duplicate subsequence matching for video streams. A simple but effective algorithm called incremental similarity update is proposed to address the problem. A similarity upper bound between two videos can be calculated incrementally by taking a lightweight computation to filter out the unnecessary time-consuming computation for the actual similarity between two videos. We integrate the algorithm with inverted frame indexing to scan video sequences for matching near-duplicate subsequences. Four state-of-the-art methods are implemented for comparison in terms of the accuracy, execution time, and memory consumption. Experimental results demonstrate the proposed algorithm yields comparable accuracy, compact memory size, and more efficient execution time.**

I. INTRODUCTION

The rapid development of multimedia technologies in recent years has spurred enormous growth in the amount of digital video content available for public access. Since such content can be easily copied, edited, and disseminated via the Internet, the proliferation of duplicate videos has become a serious problem. Near-duplicate video detection techniques have therefore generated a great deal of interest in the research community and the multimedia industry. The techniques enable content owners and managers to exploit potential video search applications, such as content protection, data mining, commercial detection, topic tracking, piracy removal, tag suggestion, search result reranking and clustering. For example, a content owner such as Disney can monitor a TV broadcast stream to see if any stream subsequence infringes the content owner's copyright.

The near-duplicate video detection techniques can be generally classified into two categories:

- **Whole sequence matching:** Given a query sequence Q and a reference sequence R, we assess their similarity. If the similarity is greater than a predefined threshold, we consider Q and R are near-duplicate. Here, Q and R must have the same frame length or the same feature dimension.
- **Partial subsequence matching:** Suppose that a subsequence $\tilde{Q} \in Q$ and a subsequence $\tilde{R} \in R$. If their similarity is greater than a predefined threshold, we consider $\tilde{Q}$ and $\tilde{R}$ are near-duplicated. It is not required that Q and R have the same frame length or the same feature dimension.

Apparently, whole sequence matching is a special case of partial subsequence matching. In this paper, we focus on partial subsequence matching. Particularly, we address the efficiency problem of subsequence matching between a continuous stream and a video database.

Most of existing stream monitoring methods do not address the efficiency issue adequately [3][4][5][9]. These methods are generally designed for using a short target sequence to scan over a long query stream efficiently. However, it is impractical to repeat the scanning procedure for each of target files in a large-scale database. Yan et al. [8] proposed an index structure to deal with multiple target files. An $m \times n$ array was built for m hash functions and n target files. Each array element stored the target file id, its min-hash values, and up/down links. Based on the index structure, they designed an index probing algorithm to select a list of target files needed to be compared. However, the index structure consumes a substantial volume of memory (consider the case where m = 1,000 and n = 100,000). It becomes too cumbersome to be efficient for large-scale retrieval.

In this study, we present a novel framework to address the efficiency issue in near-duplicate subsequence matching. The major contribution of this paper is that we propose an efficient algorithm called *incremental similarity update*. The update algorithm is executed incrementally by leveraging the prior knowledge of the previous stream/file subsequences, making the computation lightweight and efficient. Suppose that a database file is scanned by the current stream buffer. If the new incoming file frame hits the stream buffer, we then update the similarity upper bound and hit timestamp of the database file. It is guaranteed no true positive is missed.

The remainder of this paper is organized as follows. Section 2 presents the feature extraction and similarity measurement. In Section 3, we detail the subsequence matching using the incremental similarity update algorithm. Section 4 shows and discusses the experimental results compared with some baseline methods. In Section 5, we summarize our conclusion.

* *Corresponding Author*

II. Feature Extraction and Similarity Measurement

Let $Q = \{q_1, q_2, ...\}$ be a query video stream with continuous frames $q_1, q_2, ...$, $B_j = \{q_{j-m+1}, q_{-m+2}, \dots, q_j\}$ be a stream buffer containing m frames of Q being monitored, $\mathbf{D} = \{T^{(1)}, T^{(2)}, \dots, T^{(r)}\}$ be a video database with r target videos, and $T^{(i)} = \{t_1^{(i)}, t_2^{(i)}, \dots, t_n^{(i)}\}$ be the ith target video with n frames. Assume that $m < n$. The objective is to find a subsequence of $T^* \in \mathbf{D}$ that is a near-duplicate of B_j.

We adopt the ordinal relation [6] to characterize a video frame. The ordinal relation is described in the following. We divide a frame into 3×3 non-overlapping blocks and calculate each block's average intensity, as shown in Figure 1. The ordinal relation $R(s, t)$ returns a binary bit that indicates if $I(s)$ is bigger than $I(t)$, where function $I(s)$ returns the average intensity of sth block:

$$R(s,t) = \begin{cases} 1 & if\ I(s) > I(t), \\ 0 & otherwise. \end{cases}$$

We select sixteen ordinal relations as listed in Table I and transform the sixteen bits to the corresponding ordinal code $\mathbf{OC}(f) \in [0, 65535]$ for representing a frame f.

Table I. Ordinal relation.

	s	t		s	t
R_1(s, t)	(2, 2)	(1, 2)	R_9(s, t)	(1, 2)	(2, 3)
R_2(s, t)	(2, 2)	(2, 3)	R_{10}(s, t)	(2, 3)	(3, 2)
R_3(s, t)	(2, 2)	(3, 2)	R_{11}(s, t)	(3, 2)	(2, 1)
R_4(s, t)	(2, 2)	(2, 1)	R_{12}(s, t)	(2, 1)	(1, 2)
R_5(s, t)	(2, 2)	(1, 1)	R_{13}(s, t)	(1, 1)	(1, 3)
R_6(s, t)	(2, 2)	(1, 3)	R_{14}(s, t)	(1, 3)	(3, 3)
R_7(s, t)	(2, 2)	(3, 3)	R_{15}(s, t)	(3, 3)	(3, 1)
R_8(s, t)	(2, 2)	(3, 1)	R_{16}(s, t)	(3, 1)	(1, 1)

A sequence of frames V is represented as a bag-of-words (BoW) model form. Conceptually, V is a 65536-bin histogram; each bin represents an ordinal code and stores the number of the code occurring in V. The similarity between two sequences V_1 and V_2 is defined by the *dice coefficient*:

$$DICE(V_1, V_2) = \frac{2 \cdot |V_1 \cap V_2|}{|V_1| + |V_2|} = \frac{2 \cdot \sum_{b=1}^{65536} \min\left(H_b(V_1), H_b(V_2)\right)}{|V_1| + |V_2|}, \quad (1)$$

where function $H_b(V)$ returns the bth histogram bin value of V, and $|V|$ denotes the cardinality of V, i.e., $\sum_{b=1}^{65536} H_b(V)$. The use of BoW and Equation (1) has two major benefits: one is its simple computation, and the other is its high tolerance of temporal variations (e.g., frame dropping), which occur frequently in an unstable streaming environment.

III. Subsequence Matching

We construct an inverted index table to manage all target files based on their ordinal code histogram. Denote the inverted file index table as **X**, which contains 65536 cells corresponding to the ordinal codes {0, 1, 2, ... , 65535}. Each cell stores a link list of target files' identities. For the ith target file $T^{(i)}$, we insert i to the bth cell $\mathbf{X}(b)$ if $H_b(T^{(i)}) > 0$.

Given a database file that is hit by the new incoming query stream keyframe (through the inverted file index table **X**), we perform linear scanning over candidate target files to locate the subsequences that are near-duplicates of the query stream buffer. Denote the hit database file as a candidate. A conventional linear scanning process employs a sliding window to bound a subsequence of keyframes in a candidate file and computes the similarity between the window subsequence and query stream buffer. Each time the sliding window moves forward one keyframe over the candidate file for next similarity computation. Although the process is simple and effective, the exhaustive computation might be time-consuming when scanning a lengthy candidate file.

We propose the incremental similarity update algorithm to expedite the linear scanning process. Suppose that at stream timestamp j, q_j is the new incoming stream keyframe and $B_j = \{q_{j-m+1}, q_{-m+2}, \dots, q_j\}$ is the stream buffer to be monitoring currently. The objective is to accelerate the search for a candidate list of target files **G**, where $T \in \mathbf{G}$, $DICE(T, B_j) > \theta$, θ is a predefined threshold.

We initialize three attributes for each target file $T^{(i)}$:

- *maxHitCount*: the maximum number of hit keyframes between $T^{(i)}$ and B_j (set to 0 initially).
- *lastHitTime*: the last hit stream timestamp (set to 0 initially).

Denote the ordinal code of q_j as $\mathbf{C}(q_j)$. If the ith target file $T^{(i)}$ is hit by q_j, i.e., $i \in \mathbf{X}(\mathbf{C}(q_j))$, we update the attributes of $T^{(i)}$ by the incremental similarity update algorithm, as summarized in Figure 1. We calculate the minimum number of non-hit keyframes in $B_{T^{(i)}.lastHitTime}$ (x in Line 1) and the number of stream keyframes passed since the last hit (y in Line 2). We then infer the minimum number of hit keyframes in the past stream keyframes (z in Line 3) and the maximum number of hit keyframes in the current stream buffer (Line 4).

Algorithm 1: Incremental similarity update.
Input: A target file $T^{(i)}$ to be updated and stream timestamp j.
Output: The updated attributes of $T^{(i)}$.

1. $x = m - T^{(i)}.maxHitCount$.
2. $y = j - T^{(i)}.lastHitTime$.
3. $z = \max\{y - x, 0\}$.
4. $T^{(i)}.maxHitCount = \max\{T^{(i)}.maxHitCount - z, 0\} + 1$.
5. $T^{(i)}.lastHitTime = j$.

Figure 1. The incremental similarity update algorithm.

Algorithm 2 lists the pseudo code of subsequence matching, as listed in Fig. 2. Let $T^{(i)} = \left\{t_1^{(i)}, t_2^{(i)}, \dots, t_n^{(i)}\right\}$ be a candidate target file with n keyframes, and $W_k^{(i)} = \left\{t_{k-m+1}^{(i)}, t_{k-m+2}^{(i)}, \dots, t_k^{(i)}\right\} \in T^{(i)}$ be a window subsequence with m keyframes. The objective is to find a $W_k^{(i)}$, $k \in \{m, m+1, \dots, n\}$, that is a near-duplicate of stream buffer B_j. Suppose that $t_k^{(i)}$ is the new incoming keyframe when advancing the sliding window. If the ordinal codeword

of $t_k^{(i)}$ hit those of B_j, Algorithm 1 is applied to update the attributes of $T^{(i)}$ (Lines 4-5). Based on a predefined threshold φ, we use the maximum number of hit frames and the dice coefficient to verify if $W_k^{(i)}$ is a near-duplicate of B_j (Lines 6-10). Algorithm 2 is accelerated by employing the lightweight computation of the similarity upper bound as a rough filter, rather than applying the exhaustive computation of the actual similarity directly.

Algorithm 2: Near-duplicate subsequence matching.
Input: A database files C_j and the stream buffer B_j.
Output: A near-duplicate list of video sequences **ND**.

1. for each $T^{(i)} \in C_j$
2. for $k = m$ to n /* $m = |B_j|$; $n = |T^{(i)}|$ */
3. $\boldsymbol{W_k^{(i)}} = \left\{\boldsymbol{t_{k-m+1}^{(i)}}, \boldsymbol{t_{k-m+2}^{(i)}}, \ldots, \boldsymbol{t_k^{(i)}}\right\}$.
4. if $\mathrm{OC}\left(\boldsymbol{t_k^{(i)}}\right) \in \{\mathbf{OC}(\boldsymbol{q}) \mid \boldsymbol{q} \in \boldsymbol{B_j}\}$
5. *IncrementalSimilarityUpdate*($T^{(i)}$, k).
6. if $\boldsymbol{T^{(i)}.maxHitCount} > \frac{(|\boldsymbol{W_k^{(i)}}|+|\boldsymbol{B_j}|)\cdot\boldsymbol{\varphi}}{2}$
7. $\boldsymbol{sim} = \mathbf{DICE}(\boldsymbol{W_k^{(i)}}, \boldsymbol{B_j})$.
8. $\boldsymbol{T^{(i)}.maxHitCount} = \left|\boldsymbol{W_k^{(i)}} \cap \boldsymbol{B_j}\right|$.
9. if $sim > \varphi$
10. Insert a 3-tuple <B_j, $T^{(i)}$, k, sim> to **ND**.

Figure 2. The incremental similarity update algorithm.

IV. Experimental Results

In this section, we evaluate the performance of the proposed algorithms in terms of accuracy, execution time, and memory consumption. The programs were implemented in C++, and the testing environment was carried out on Window 7, Core i7 3.4GHz quad core CPU (only one thread was used), and 16GB memory.

We collected videos from three sources. The first source was the CC_WEB_VIDEO dataset [1], which contained 24 folders of video clips, totally 12,890 files and approximately 732 hours. The second source was the TRECVID 2011 IACC.1.A dataset [7], which contained 8,175 files and approximately 225 hours. The last source was the YouTube videos. The most popular 135 queries that were selected from Google Zeitgeist were used to search and download YouTube videos; 50,715 Web videos and approximately 3,571 hours were crawled in total. We compiled the videos of three sources into a *target database* containing 71,780 files and approximately 4,528 hours.

A number of *query streams* were compiled as follows. We prepared an unrelated video with 1,710 seconds that did not contain any near-duplicated content to the target database. From each folder of the CC_WEB_VIDEO dataset, the first video file, denoted as the seed, was selected and inserted in the unrelated video to replace the original content at random. In total 24 query streams were created to search for near-duplicates in the target database. They were transcoded into the MP4 file format by the H.264 codec, which was widely used in internet streaming and TV broadcasting. The ground truth was labeled manually for each query stream; true positives were almost in the corresponding folder of the CC_WEB_VIDEO dataset.

Suppose the candidate database file list of a stream buffer is generated through the inverted file index table **X**. In the subsequence matching phase, we scan each candidate target file to locate the subsequences that are near-duplicates of the stream buffer. If the detected target subsequence and the stream buffer have any overlap content, they are considered corrected and denoted as the true positives.

We evaluated the accuracy by the mean average precision (MAP) metric. The average precision (AP) is defined by [2]:

$$AP = \frac{1}{n}\sum_{i=1}^{n}\frac{i}{r_i},$$

where n is the number of relevant videos to the query stream seed, and r_i is the rank of the ith retrieved relevant video. We set the parameters of stream buffer size m = 30 and threshold φ = 0.1. The evaluation metrics include PR curves, localization error, and matching time in average of the 24 query seed videos (i.e., stream buffers).

Four subsequence locating methods proposed by Kashino et al. [4], Huang et al. [3], Zhou and Chen [9], and Liu et al. [5], were implemented as the baselines. Figure 3 shows the average PR curves of the 24 query stream buffers. Table II summarizes the experimental results in terms of MAP, matching time, and memory consumption. Notations HP, FS, EMD, and GPM represent the methods of histogram pruning (Kashino et al. [4]), frame skipping (Huang et al. [3]), Earth mover's distance (Zhou and Chen [9]), and Gaussian probability model (Liu et al. [5]), respectively. The stream buffer size m and the similarity threshold φ used in ISU, HP, and FS were all set to 30 and 0.1, respectively.

Our method (ISU) and Kashino et al.'s method (HP) achieved the best accuracy among these methods. The matching time of ISU is slightly faster than that of HP. The main reason is that when scanning over a video, ISU skips a more number of keyframes without computing the time-consuming actual similarity. The number of skipped keyframes for HP is proportional to the difference between the actual similarity and the threshold. The difference may be changed dramatically. Moreover, if the difference is getting small, e.g., using a lower threshold, HP becomes less efficient. ISU yields a relatively stable matching time.

Huang et al.'s method (FS) yielded a slightly worse accuracy than ISU and HP. The number of skipped keyframes of FS is more than that of HP. This is because FS combines HP and temporal order checking, so that a more number of keyframes can be skipped without applying the dynamic time warping algorithm. However, the dynamic time warping algorithm is a relatively time-consuming process compared with the set intersection operation. Consequently FS spends a higher computation cost than HP. Note that temporal order checking and dynamic time warping are both based on the assumption that near-duplicate content are of the similar temporal order. While employing the temporal information can enhance the discriminative power of a video, its tolerance

to temporal editing (such as randomly insertion/deletion and severely frame dropping) is degenerated meanwhile.

Zhou and Chen's method (EMD) extracted the complicated video cuboid signature representation and calculated the time-consuming Earth mover's distance to overcome severe video transformations. Although EMD employs the spatial and temporal locality of adjacent keyframes to accelerate the sequence scanning process, we consider the overall computation complexity is too high to be suitable for real-time stream monitoring in a large-scale video database.

Liu et al.'s method (GPM) took the least matching time to locate near-duplicates by employing the inverted keyframe index table of each target file. Rather than sequence scanning, only a number of keyframe lists indexed from the stream buffer are searched to find their intersection. However, GPM assumes that no frame gets lost between two near-duplicated videos. This assumption is seldom held in a stream video; frame loss occurs frequently and thus degrades the accuracy.

The memory consumption of EMD is much larger (about 47 times) than that of the other four methods. EMD uses a number of complicated video cubiod descriptors to represent a keyframe, while the other four methods use the same feature representation, i.e., each keyframe is represented as an ordinal codeword. The tremendous amount of the cuboid data volume hinders EMD to be efficient in a large-scale database.

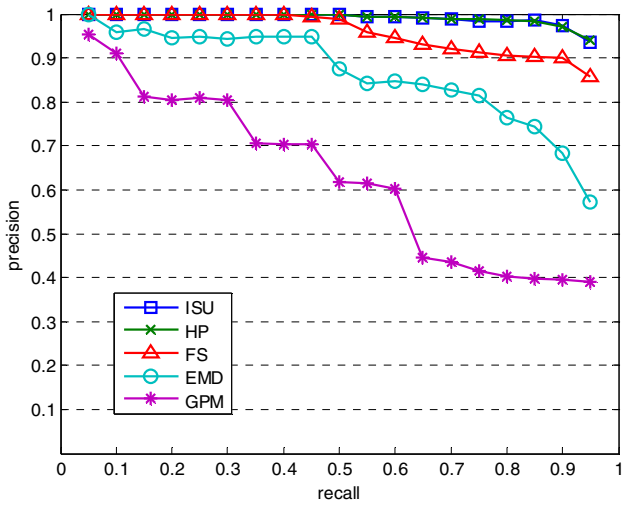

Figure 3. The PR curves of ISU, HP, FS, EMD, and GPM for subsequence locating.

TABLE II. COMPARISONS ON MAP, MATCHING TIME, AND MEMORY CONSUMPTION FOR FILE MATCHING

	MAP	Matching Time (ms)	Memory (MB)
ISU	0.993	6.73	34.70
HP	0.993	8.50	34.70
FS	0.971	28.00	34.70
EMD	0.923	8502.00	1621.94
GPM	0.782	0.90	34.70

V. CONCLUSIONS

To address the efficiency problem of near-duplicate subsequence matching for streams, we propose the incremental similarity update algorithm that calculates the similarity upper bound between two sequences incrementally and reduce unnecessary computation for the actual but complex similarity. Substantial experimental results demonstrate that the proposed algorithm can successfully yield a satisfactory accuracy, efficient execution time, and compact memory consumption.

ACKNOWLEDGMENT

This work was supported in part by the National Science Council of Taiwan under Grants NSC NSC101-2221-E-415-023.

REFERENCES

[1] CC_WEB_VIDEO: Near-Duplicate Web Video Dataset. http://vireo.cs.cityu.edu.hk/webvideo/

[2] W. Dong, Z. Wang, M. Charikar, and K. Li, "Efficiently matching sets of features whti random histograms," In *Proceedings of ACM International Conference on Multimedia (MM)*, Vancouver, Canada, Oct. 26-31, 2008.

[3] Z. Huang, H. T. Shen, J. Shao, B. Cui, and X. Zhou, "Practical online near-duplicate subsequence detection for continuous video streams," *IEEE Transactions on Multimedia*, Vol. 12, No. 5, pp. 386-398, 2010.

[4] K. Kashino, T. Kurozumi, and H. Murase, "A quick search method for audio and video signals based on histogram pruning," *IEEE Transactions on Multimedia*, Vol. 5, No. 3, pp. 348-357, 2003.

[5] B. Liu, Z. Li, L. Yang, M. Wang, and X. Tian, "Real-time video copy-location detection in large-scale repositories," *IEEE Multimedia*, Vol. 18, No. 3, pp. 22-31, 2011.

[6] L. Shang, L. Yang, F. Wang, K. P. Chan, and X. S. Hua, "Real time large scale near-duplicate web video retrieval," In *Proceedings of ACM International Conference on Multimedia (MM)*, Firenze, Italy, Oct. 25-29, 2010.

[7] TRECVID 2011 Guidelines. http://www-nlpir.nist.gov/projects/tv2011/

[8] Y. Yan, B. C. Ooi, and A. Zhou, "Continuous content-based copy detection over streaming videos," *IEEE International Conference on Data Engineering (ICDE)*, Cancun, Mexico, Apr. 7-12, 2008.

[9] X. Zhou and L. Chen, "Monitoring near duplicates over video streams," In *Proceedings of ACM International Conference on Multimedia (MM)*, Firenze, Italy, Oct. 25-29, 2010.