PERGAMON

Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Information Systems 29 (2004) 641–652

www.elsevier.com/locate/infosys

Trie for similarity matching in large video databases[☆], [☆☆]

Sanghyun Park[a,*], Ki-Ho Hyun[b]

[a] *Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH), Pohang, South Korea*
[b] *School of Computer and Information Engineering, YoungSan University, South Korea*

Received 4 November 2002; received in revised form 1 May 2003; accepted 1 May 2003

Abstract

Similarity matching in video databases is of growing importance in many new applications such as video clustering and digital video libraries. In order to provide efficient access to relevant data in large databases, there have been many research efforts in video indexing with diverse spatial and temporal features. However, most of the previous works relied on sequential matching methods or memory-based inverted file techniques, thus making them unsuitable for a large volume of video databases. In order to resolve this problem, this paper proposes an effective and scalable indexing technique using a *trie*, originally proposed for string matching, as an index structure. For building an index, we convert each frame into a symbol sequence using a *window order heuristic* and build a disk-resident trie from a set of symbol sequences. For query processing, we perform a depth-first traversal on the trie and execute a *temporal segmentation*. To verify the superiority of our approach, we perform several experiments with real and synthetic data sets. The results reveal that our approach consistently outperforms the sequential scan method, and the performance gain is maintained even with a large volume of video databases.
© 2003 Elsevier Ltd. All rights reserved.

Keywords: Indexing technique; Similarity matching; Video databases; Trie; Window order heuristic; Window-based feature

1. Introduction

In recent years, the growth of the amount of video data in multimedia databases has been tremendous. Similarity matching in video databases is of growing importance in many new applications such as *video clustering* and *digital video libraries*. In order to provide effective and efficient access to relevant data in large databases, there have been many research efforts [1–8] in indexing of video databases with features such as colors, textures, semantics, and motions.

A large number of previous video indexing approaches depend on the *sequential scan* method to retrieve similar videos in a database. Mohan [9] and Vailaya et al. [10] proposed video sequence matching methods using action similarity and the combination of image content and image motion, respectively. Lienhart et al. [11] and Sanchez et al. [12] used color coherence vectors and principal

[☆]Recommended by Dr. Felipe Cariño. Jr. (fcarino@pacbell.net).
[☆☆]This work was supported in part by the Brain Korea 21 Project in 2003.
*Corresponding author.
E-mail address: sanghyun@postech.ac.kr (S. Park).

0306-4379/03/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0306-4379(03)00046-2

components of color histograms, respectively, for detecting similar commercial video clips. Adjeroh et al. [13] employed the *vstring* representation for video sequences and introduced *vstring edit distance* as a similarity indicator. Ardizzone et al. [3] extracted a single MPEG motion vector from each 16×16 sub-image, and Meng and Chang [6] used low-level motion features such as zoom and pan of camera. Since all the works mentioned above use the sequential scan method, their performance deteriorates when the database is large.

Squire et al. [14] proposed the use of inverted file techniques for feature-based image retrieval, and Hampapur et al. [8] applied the inverted file techniques to media tracking. Since both approaches keep the inverted files in main-memory during the query processing, they are not suitable for a large volume of video databases.

In the area of string matching, a trie [15] has been extensively used as an index structure to find the strings that are exactly or approximately matched to a given query string. A trie is also an efficient index structure for both multimedia and time-series data because it does not have the *dimensionality curse problem*. Merrett et al. [16] employed a trie for indexing a set of spatial data objects and Park et al. [17,18] proposed to use a *suffix tree* [15], a trie constructed from the suffixes of strings, for subsequence matching problem.

In this paper, we propose an effective and scalable indexing technique for similarity matching in large video databases using a trie as an index structure. Assuming that all video frames are divided by the same number of windows, we do the following to build an index: (1) extract a feature from each window and convert it into a discrete symbol, (2) convert each frame into a symbol sequence using a *window order heuristic*, and (3) build a disk-resident trie from a set of symbol sequences. For query processing, we do the following: (1) take the first two steps of index construction to generate a symbol sequence for each target frame, (2) find the data frames closest to each target frame by performing a depth-first traversal on the trie, and (3) execute a *temporal segmentation* to find the videos or sub-videos similar to a target video.

This paper is organized as follows. Section 2 explains the two features employed in this paper, and Section 3 describes our index construction and query processing methods. Section 4 evaluates the proposed approach through experiments in comparison with the sequential scan method. Finally, Section 5 summarizes and concludes the paper.

2. Feature extraction

Since we divide every frame into multiple windows, we take window-based feature extraction methods. This section describes the two window-based features, a motion feature and an ordinal feature, employed in this paper. We assume that all the data videos are already digitized before coming into the indexing stage. The detailed description on the video digitization process, manipulation of video source format, and the video compression standards can be found in [19].

2.1. Motion feature

Motion [20,21] is an essential feature of video sequences. By analyzing the motions embedded in a video sequence, we can extract its unique temporal information. The algorithms for estimating the displacement vectors of moving objects can be roughly classified into the three groups such as the optical-flow-based approach, the feature-based approach, and the window-matching-based approach. We employ the window-matching-based approach since it is well suited to our indexing method.

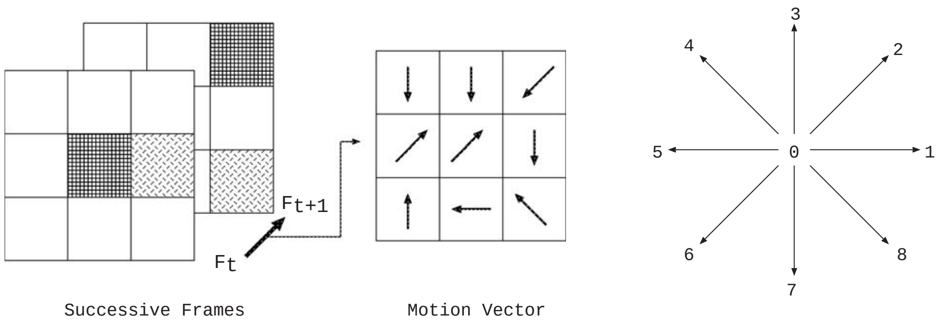
Let F_t and F_{t+1} be the current frame and its subsequent frame, respectively. The frames are partitioned into multiple windows and a motion vector is estimated for each window. To find the displacement in F_{t+1} for a point (x, y) in F_t, the surrounding window of the point is compared to the pixels in the search area. The best match is found according to the criterion of the minimum mean-square error. Then, the direction of the motion vector is quantized into q directions or levels. In addition to the q directions, if the motion vector of a window has zero magnitude, the window can also be assigned a level 0. Fig. 1 shows the steps of the motion estimation process.

2.2. *Ordinal feature*

We employ the ordinal measure [9,22] to capture the spatial information of a video sequence. The ordinal variable is drawn from a discrete ordered set like the grades in schools. The ratio between two measurements is not of consequence; only their relative ordering is relevant. The relative ordering between measurements is expressed by their ranks. A rank permutation is obtained by sorting the sample in ascending order and labeling them using integers $[1, 2, 3.....m]$, m being the size of the sample. The ordinal measure has the advantage that it is less sensitive to a pixel variation. In our application, the average gray level in each window is viewed as an ordinal variable. The set of average intensities is sorted in ascending order and the rank is assigned to each window of a frame. Fig. 2 shows an example.

3. Indexing method

This section defines the problem we are going to solve, and describes our index construction and query processing methods.



(a) Motion vector estimation (b) Quantization of motion vector

Fig. 1. Motion estimation process.

25.3	80.2	89.8
152.3	93.2	125.7
160.3	225.3	230.6

(a) average intensity value of each window in a frame

1	2	3
6	4	5
7	8	9

(b) ordinal features in a frame

Fig. 2. Example of ordinal measure.

3.1. Problem definition

Let us first explain the notations and the similarity measures used in this paper. $V = \langle V[1], \ldots, V[n] \rangle$ denotes a video with n frames and $F = \langle F[1], \ldots, F[m] \rangle$ denotes a frame with m windows. The distance of any two frames is defined as the number of window pairs whose distances are beyond a noise threshold. More specifically, given a distance or noise threshold ε, the distance of any two frames F_i and F_j with m windows is defined as the number of window pairs $(F_i[h], F_j[h])$ $(1 \leqslant h \leqslant m)$ satisfying the inequality $|F_i[h] - F_j[h]| > \varepsilon$. The distance of F_i and F_j becomes 0 when every window in F_i is within ε from the corresponding window in F_j and becomes m when the distance of every window pair is larger than ε.

Given a target video $V_t = \langle V_t[1], \ldots, V_t[n_t] \rangle$ and a data (sub-) video $V_d = \langle V_d[1], \ldots, V_d[n_d] \rangle$, their similarity is defined as the number k of such frame pairs $(V_t[i_h], V_d[j_h])$ $(1 \leqslant h \leqslant k,\ 1 \leqslant i_h \leqslant n_t,\ 1 \leqslant j_h \leqslant n_d)$ that (1) $V_d[j_h]$ is the data frame most similar to the target frame $V_t[i_h]$, and (2) they preserve the temporal coherence [8] (that is, $i_{h-1} \leqslant i_h$ and $j_{h-1} \leqslant j_h$).

The proposed similarity measure is exactly same as the one proposed by Hampapur et al. [8] except for a noise threshold ε used in the definition of frame distance. Our similarity measure enables accurate video matching even when videos have different frame rates, and gives end users more flexible control over frame similarity.

Then, the problem we are going to solve is defined as follows: Given a target video V_t, a noise threshold ε, and a set of data videos stored in a database, we want to find the k data sub-videos $V_d[p:q]$ most similar to V_t. Here, $V_d[p:q]$ represents the sub-video of V_{d^C} containing the frames in positions p through q.

3.2. Index construction

Our indexing approach uses a trie, which was originally proposed for string matching. A *trie* is a tree structure used for indexing a set of keywords. The internal nodes are empty and the edges store symbols. A path from the root to a leaf represents a symbol sequence inserted into a trie. Usually the leaf nodes store the identifiers of symbol sequences. A trie holds all the information contained in the original symbol sequences but common prefixes are stored only once. Thus, there is a large potential for compression if symbol sequences contain a large number of common prefixes and the lengths of common prefixes are long. Fig. 3(a) shows the trie constructed from two symbol sequences $X = \langle A, B, B, B \rangle$ and $Y = \langle A, A, B, B \rangle$, and Fig. 3(b)

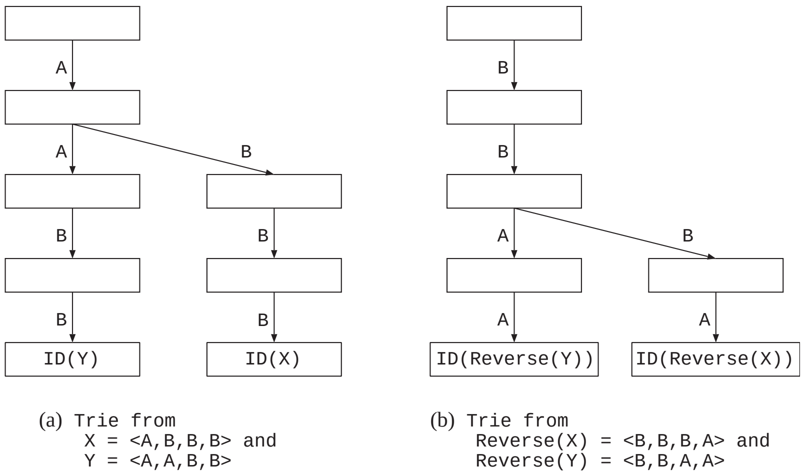

A
A
B
B
B
B
B
ID(Y)
ID(X)
(a) Trie from X = <A,B,B,B> and Y = <A,A,B,B>
B
B
A
B
A
A
ID(Reverse(Y))
ID(Reverse(X))
(b) Trie from Reverse(X) = <B,B,B,A> and Reverse(Y) = <B,B,A,A>

Fig. 3. Example tries.

shows the trie from $Reverse(X) = \langle B, B, B, A \rangle$ and $Reverse(Y) = \langle B, B, A, A \rangle$.

Now let us describe our index construction method. As shown in Fig. 4, it consists of the three steps: pre-processing, symbol sequence generation, and trie construction.

3.2.1. Pre-processing

The pre-processing steps of an index construction include windowing, feature extraction, and symbolization.

Step 1 (*windowing*): We divide each frame into a fixed number (for example, 225) of windows. It is not required that all windows have the same size. For example, if we know that the center area of a frame is more important than the other areas, then we may generate more windows in the center area. That is, the windows in the center area may become smaller than those in the other areas.

Step 2 (*feature extraction*): From each window, we compute a motion feature or an ordinal feature explained in Section 2.

Step 3 (*symbolization*): We convert each motion or ordinal feature into the corresponding symbol.

3.2.2. Symbol sequence generation

To generate a symbol sequence for each frame, this step reads the windows of a frame in the predefined order. Notice that the trie becomes more compact when symbol sequences contain more and longer common prefixes. For example, let us compare the two tries in Fig. 3. Since the common prefix of $Reverse(X)$ and $Reverse(Y)$ is longer than that of X and Y, the trie from $Reverse(X)$ and $Reverse(Y)$ has fewer nodes.

To maximize the number and the length of common prefixes, we use the following heuristic: (1) for each window, we determine the dominant symbol and compute its occurrence ratio, and (2)

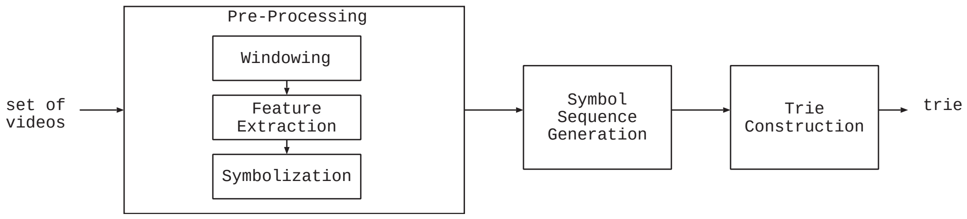
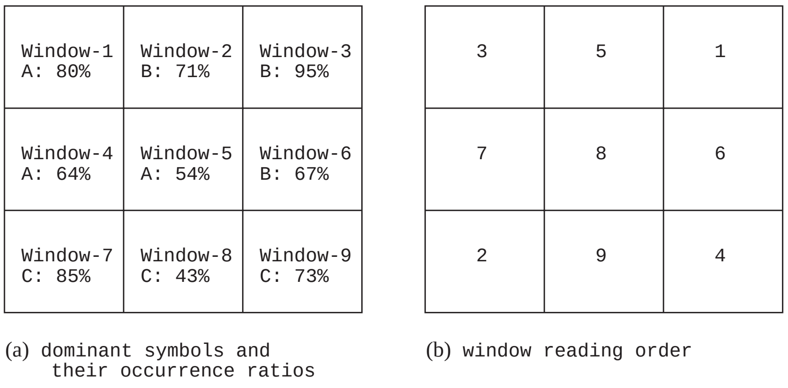


Fig. 4. Index construction steps.



Window-1 A: 80%	Window-2 B: 71%	Window-3 B: 95%
Window-4 A: 64%	Window-5 A: 54%	Window-6 B: 67%
Window-7 C: 85%	Window-8 C: 43%	Window-9 C: 73%

(a) dominant symbols and their occurrence ratios

3	5	1
7	8	6
2	9	4

(b) window reading order

Fig. 5. Window reading order with nine windows.

we rank the windows according to the occurrence ratios of their dominant symbols. Then the window reading order is determined by this ranking order. We call this heuristic a *window order heuristic*. Fig. 5(a) shows nine windows with their dominant symbols and their occurrence ratios, and Fig. 5(b) shows the window reading order determined by our heuristic.

3.2.3. Trie construction

This step makes an entry for each frame and inserts it into a trie. The index entry for the *p*th frame of a video V_i is denoted as (symbol-sequence, i, p). Since every symbol sequence has the same length, we can easily take the disk-based trie construction method: (1) sort index entries according to the ascending order of their symbol sequences, and (2) append each index entry into the trie under construction. This method keeps only the index entry being inserted and the last node of each level inside the main memory, thus making our approach suitable for a large volume of video databases.

As an example, consider the first two frames of a video V_1 and the first three frames of a video V_2. Index entries are appended according to the ascending order of their symbol sequences: $V_2[3] \rightarrow V_2[1] \rightarrow V_1[1] \rightarrow V_1[2] \rightarrow V_2[2]$. Fig. 6 shows the trie constructed from this data set.

- $V_1[1] = \langle C, C, B, B, C, A, A, B, B \rangle$.
- $V_1[2] = \langle C, C, C, B, A, C, B, B, A \rangle$.
- $V_2[1] = \langle A, A, C, C, A, B, B, C, C \rangle$.
- $V_2[2] = \langle C, C, C, C, A, A, B, B, B \rangle$.
- $V_2[3] = \langle A, A, A, A, C, B, B, B, C \rangle$.

Since it is difficult to perform systematic *disk paging* on a trie index, a trie has been considered as a main memory data structure. However, disk paging on a trie index can be systematic if we exploit the fact that every index entry (or key) has the same length in our setting and keys are inserted into a trie according to their alphabetical orders. More specifically, we slice the trie into layers of k levels each, and then chop each layer into pages of subtries. Since the trie grows along only one direction, we can easily group the set of nodes in the same layer to generate a new page. When a page is guaranteed not to be touched any more during index construction, it can be written onto the disk safely. To locate the child page directly from its parent page during index traversal, each page contains the counters for the number of edges into and out of the page layer. Our trie paging method extends the algorithm proposed in [23]

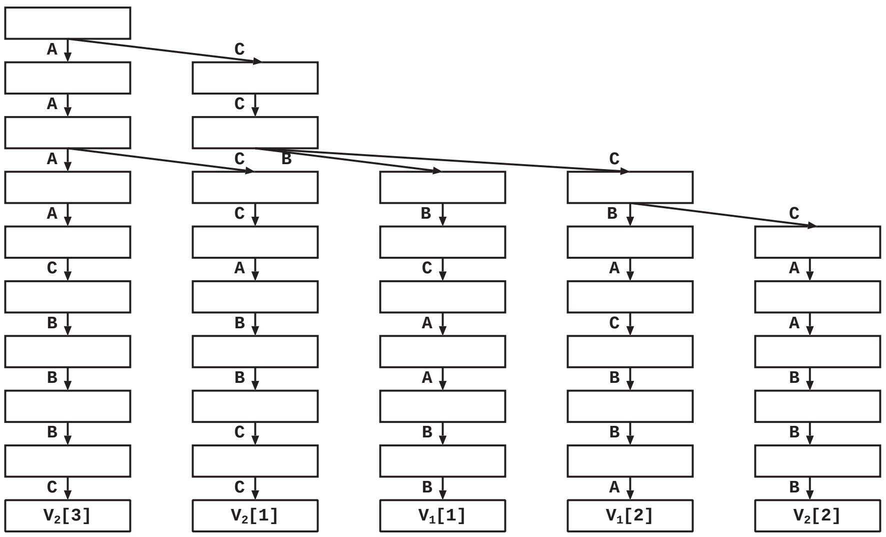


Fig. 6. Trie from the sample data set.

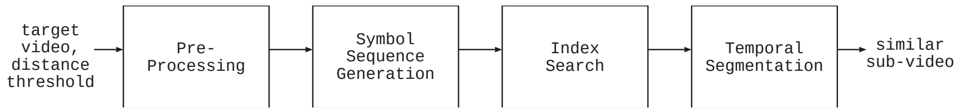


Fig. 7. Query processing steps.

where each node of a trie is assumed to have at most two child nodes.

3.3. Query processing

When a target video is submitted with a noise threshold ε, we find the similar sub-videos by taking the four steps shown in Fig. 7: pre-processing, symbol sequence generation, index search, and temporal segmentation. During the first two steps, we divide each target frame into the fixed number of windows, compute a feature from each window, and convert the feature into the corresponding symbol.

3.3.1. Index search

For each target frame which is now represented as a symbol sequence, we perform a depth-first traversal on the trie to find the best match which has the largest number of matching symbols with a target frame. Two symbols are matching when their difference is within a noise threshold ε.

The index search algorithm maintains two global variables, *BestMatch* and *MinNotMatch*. *BestMatch* stores the identifiers of the best-matching frames found so far and *MinNotMatch* keeps the number of not-matching symbols of *BestMatch*. Whenever we visit the node N, we inspect the symbol between the node N and its parent node. If that symbol is beyond the noise threshold ε from the corresponding symbol of the target symbol sequence, we increase the number of non-matching symbols $NotMatch(N)$ of the node N. If $NotMatch(N)$ is still less than *MinNotMatch*, then we further go down the trie to inspect the children nodes of N. Otherwise, we visit the sibling nodes of N. When we reach the leaf node L whose $NotMatch(L)$ is less than *MinNotMatch*, we replace *BestMatch* with the identifiers stored in the leaf node L and *MinNotMatch* with $NotMatch(L)$. Since leaf nodes may store multiple identifiers and different paths may have the same number of non-matching symbols, we may have multiple best matches for each target frame.

The proposed index search algorithm is shown in Algorithms 1 and 2. Here, $symbol(N, CN_i)$ is the symbol on the edge connecting N to its child CN_i.

```
Algorithm 1: Index search algorithm

Input: trie $T$, target frame $F_t$, noise threshold $\varepsilon$
Output: BestMatch

MinNotMatch := $\infty$;
BestMatch := {};
WindowPosition := 1;
NotMatch := 0;
VisitNode(T.rootNode, $F_t$, WindowPosition, $\varepsilon$, NotMatch);
return BestMatch;
```

3.3.2. Temporal segmentation

Whenever we find the best match for each target frame, we push it into one or more stacks according to its video and frame numbers. Each stack is labeled with a video number and each element of a stack is labeled with a frame number. For each best match $V_i[p]$, the temporal segmentation step does the following: (1) find the set of stacks labeled with i, and (2) push p onto the set of stacks whose top elements are not larger than p. After finishing this temporal segmentation, we take the k stacks with the most elements.

Suppose that a target video V_t has the five frames whose best matches are: $V_1[1]$, $V_2[10]$ for $V_t[1]$, $V_2[5]$ and $V_3[1]$ for $V_t[2]$, $V_2[7]$ and $V_4[7]$ for $V_t[3]$, $V_2[8]$ for $V_t[4]$, and $V_2[9]$ for $V_t[5]$. Fig. 8 shows the five stacks after temporal segmentation. Since the second stack is taller than any other stacks, it contains the sub-video most similar to a target video. The starting and ending offsets of the sub-video is obtained by the bottom and top

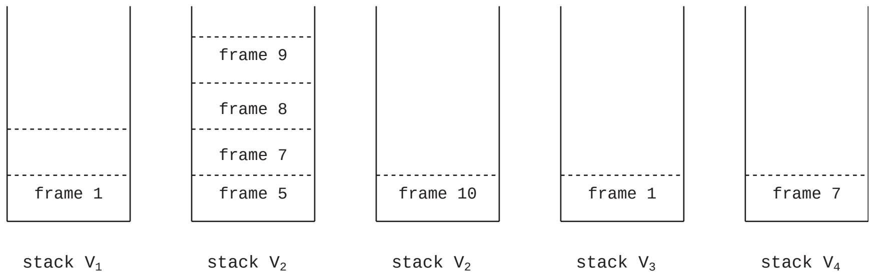


Fig. 8. Five stacks for temporal segmentation.

element values. That is, $V_2[5:9]$ is the most similar answer.

Algorithm 2: Trie traversal algorithm VisitNode

```
Input : trie node $N$, target frame $F_t$, window position $p$, noise threshold $\varepsilon$,
        number of not-matching symbols $NotMatch$
$CN \leftarrow$ GetChildren($N$);
for $i \leftarrow 1$ to $|CN|$ do
    dist := $|F_t[p] - symbol(N, CN_i)|$;
    if $dist \leq \varepsilon$ then
        if $CN_i$ is a leaf node then
            if $NotMatch == MinNotMatch$ then
                BestMatch := BestMatch ∪ {Identifier($CN_i$)};
            else
                if $NotMatch < MinNotMatch$ then
                    BestMatch := {Identifier($CN_i$)};
                    MinNotMatch := NotMatch;
        else
            VisitNode($CN_i$, $F_t$, $p+1$, $\varepsilon$, $NotMatch$);
    else
        NotMatch++;
        if $NotMatch \leq MinNotMatch$ then
            if $CN_i$ is a leaf node then
                if $NotMatch == MinNotMatch$ then
                    BestMatch := BestMatch ∪ {Identifier($CN_i$)};
                else
                    BestMatch := {Identifier($CN_i$)};
                    MinNotMatch := NotMatch;
            else
                VisitNode($CN_i$, $F_t$, $p+1$, $\varepsilon$, NotMatch);
return;
```

4. Performance evaluation

This section presents the experimental results for performance evaluation of the proposed approach. Section 4.1 describes the experimental environment and Section 4.2 shows and analyzes the experimental results.

4.1. Experimental setup

Two kinds of data videos were used for experiments: synthetic and real data videos. Synthetic videos were composed from synthetic frames whose features were taken uniformly from a finite set of integer values. The number of synthetic data videos and their average number of frames were decided according to the purpose of each experiment. As a real-life example, we used three data movies easily available on the web. The first movie was SW: *Star Wars Episode IV* that is an action movie with rapid scene changes, and the second one was SAM: *I am Sam* that is a human drama without much action. The last one was ICE: *Ice Age* that is an animation movie for children. We generated 39 sub-videos, 43 sub-videos, and 26 sub-videos from the movies SW, SAM, and ICE, respectively, and then extracted 540 key frames from each sub-video. Table 1 shows the detailed information about the real data movies used in the experiments.

For each experiment, we performed 100 queries with the target videos generated as follows: (1) select one randomly from data videos, (2) extract a sub-video from the selected data video, (3) take a random value from an appropriate range for each window of a frame, and (4) add the value to its corresponding window as a noise. We use an average elapsed time of processing 100 queries as a performance measure. To avoid buffering effect, we cleaned up the main memory right before executing each query.

The hardware platform for the experiments was the IBM Personal Computer 300PL—Pentium III

662 MHz system equipped with 256 MB RAM, and the software platform was Windows NT 4.0.

For performance evaluation, we compared our approach with the sequential scan method. The sequential scan method reads data videos sequentially to find the best matches for each target frame and then performs the temporal segmentation explained in Section 3. Notice that the set of answers retrieved by our indexing method is always identical to the set of answers obtained by the sequential scan method. This is because the sequential scan method measures a similarity of any two frames using digitized ordinal or motion features from which the trie index structure has been built.

Table 2 summarizes the parameters used in each experiment where (1) *numWins* denotes the number of windows in a frame, (2) *numSyms* is the total number of quantization symbols inside a window, (3) *noiThres* is the noise threshold given by a user (4) *numVids* is the total number of videos stored in the database, (5) *lenVids* is the average number of frames in data videos, and *lenQrys* is the number of frames in a target video.

4.2. Results and analysis

Using the ordinal features of the real data videos, Experiment 1 compares the elapsed times of our approach and the sequential scan with the increasing noise threshold ε from 0 to 4. Experiment 2 does the same thing using the motion feature. As shown in Figs. 9 and 10, our approach is about two times faster when using ordinal features and about three times faster when using motion features. Notice that our approach performs better when ε is large. This is because a large ε makes *MinNotMatch* reach its final value quickly and thus reduces the portion of a trie touched by index search.

Using a synthetic data set, Experiments 3 and 4 compare the elapsed times of the two approaches with the increasing number of windows from 9 to 225 and with the increasing number of quantization symbols from 4 to 32. As shown in Fig. 11, both approaches require more query processing time as the number of windows grows, but the slope of our approach is smaller than that of the sequential scan. As shown in Fig. 12, both

Table 1
The detailed information about the real data movies used in the experiments

	SW	SAM	ICE	*SW + SAM + ICE*
Length	2 h	2 h 12 min	1 h 21 min	5 h 32 min
Format	MPEG1	AVI	AVI	—
Size	1.16 GB	1.44 GB	0.73 GB	3.33 GB
Number of sub-videos	39	43	26	108

Table 2
Parameter values for each experiment

	Exp 1	Exp 2	Exp 3	Exp 4	Exp 5	Exp 6
Data set	*SW + SAM + ICE*	*SW + SAM + ICE*	Synthetic	Synthetic	Synthetic	Synthetic
Feature	Ordinal	Motion	Uniform	Uniform	Uniform	Uniform
numWins	25	225	9-225	16	16	16
numSyms	25	9	8	4-32	8	8
noiThres	0-4	0-3	1	1	1	1
numVids	108	108	100	100	100-400	100
lenVids	540	540	200	200	200	200-1,000
lenQrys	30	30	20	20	20	20
index size	3.9M	27.2M	0.7-9.3M	0.9-1.0M	0.9-3.8M	0.9-4.7M

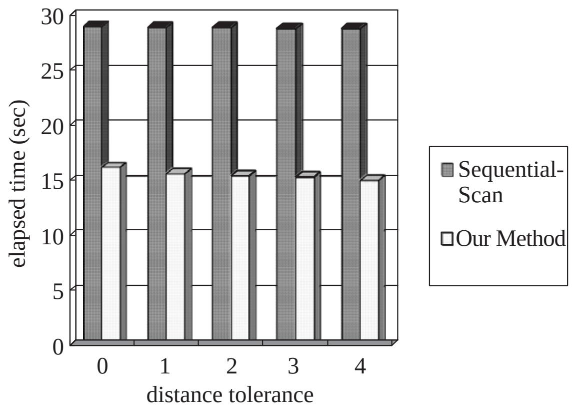


Fig. 9. Comparison of elapsed times with increasing noise threshold ($SW + SAM + ICE$ with ordinal features).

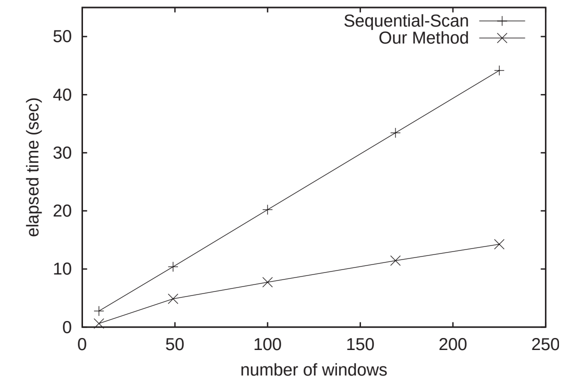


Fig. 11. Comparison of elapsed times with increasing number of windows (synthetic data set).

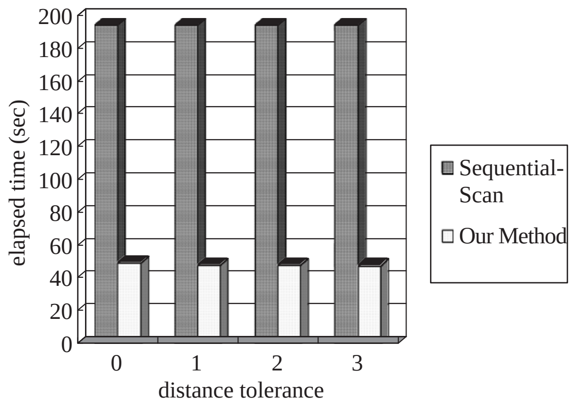


Fig. 10. Comparison of the elapsed times with the increasing noise threshold ($SW + SAM + ICE$ with motion features).

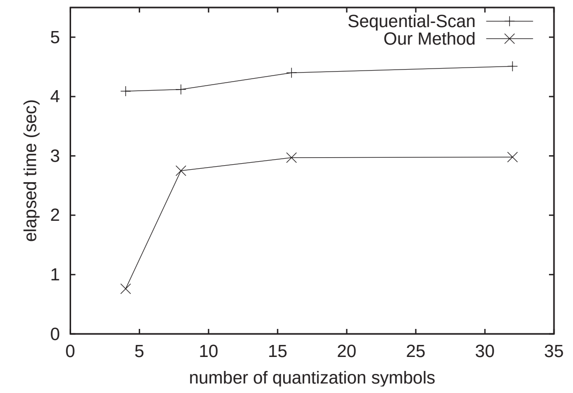


Fig. 12. Comparison of the elapsed times with the increasing number of quantization symbols (synthetic data set).

approaches show the slight increase of query processing time although our approach has sharp increasing pattern when the number of quantization symbols is between 5 and 10. This verifies that the number of common subsequences and their average length reduce as the number of quantization symbols increases. However, the slope becomes steady after the number of symbols exceeds a certain threshold.

Using a synthetic data set, Experiments 5 and 6 compare the elapsed times of the two approaches with the increasing number of videos from 100 to 400 and with the increasing average number of

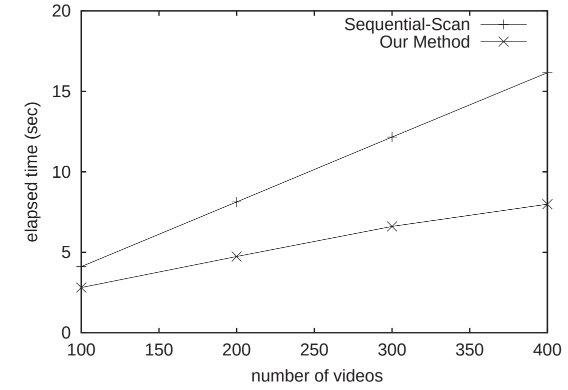


Fig. 13. Comparison of the elapsed times with the increasing number of videos (synthetic data set).

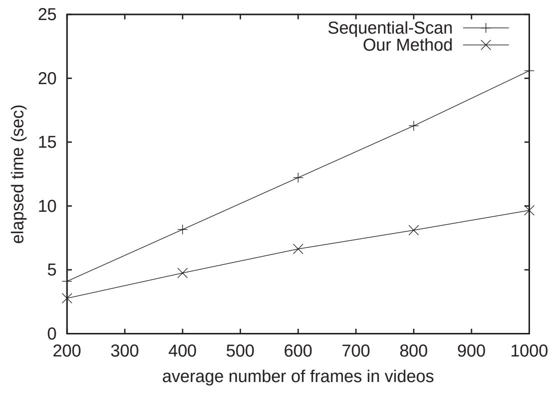


Fig. 14. Comparison of the elapsed times with the increasing average number of frames in videos (synthetic data set).

frames in videos from 200 to 1,000. Figs. 13 and 14 show the experimental results. Due to the increasing size of video databases, both approaches become slow when the number of videos and their average number frames increase. However, the slope of our approach is smaller than that of the sequential scan. Therefore, our approach is more scalable and better suited for large video databases.

5. Conclusion

Similarity matching in video databases is of growing importance in many new applications. Although there have been many research efforts for providing efficient access to relevant data in video databases, most of the previous works relied on sequential matching methods or memory-based inverted file techniques, thus making them unsuitable for a large volume of video databases.

In order to resolve this problem, this paper proposed an efficient and scalable indexing technique using a trie as an index structure. Our contributions are: (1) adapt a trie for video indexing, (2) introduce a window order heuristic for reducing the index size and thus improving the search performance, and (3) propose effective index search and temporal segmentation algorithms. To verify effectiveness and scalability of our approach, we performed the experiments with real and synthetic video databases. The experimental results reveal that our approach consistently outperforms the sequential scan method, and the performance gain is maintained even with a large volume of video data.

There are lots of applications whose main operations are similarity-based searching in video databases. Content-based video clustering, content-based video copy detection, and video on demand (VOD) are such applications, just to name a few. The viability of these applications heavily depends on the ability to retrieve data *efficiently* and *accurately* from video databases. These applications can benefit from the proposed approach due to its effectiveness and scalability.

References

[1] R. Tusch, H. Kosch, L. Boszormenyi, VIDEX: an integrated generic video indexing approach, in: Proceedings of ACM Multimedia, Los Angeles, CA, USA, 2000, pp. 448–451.

[2] S. Dagtas, A. Ghafoor, Indexing and retrieval of video based on spatial relation sequences, in: Proceedings of ACM Multimedia, Vol. 2, Orlando, FL, USA, 1999, pp. 119–122.

[3] E. Ardizzone, M.L. Cascia, A. Avanzato, A. Bruna, Video indexing using MPEG motion compensation vectors, in: Proceedings of the IEEE International Conference on Multimedia Computing System, Vol. 2, Florence, Italy, 1999, pp. 725–729.

[4] J. Wei, Z.-N. Li, I. Gertner, A novel motion-based active video indexing method, in: Proceedings of the IEEE International Conference on Multimedia Computing System, Vol. 2, Florence, Italy, 1999, pp. 460–465.

[5] E. Sahouria, A. Zakhor, Motion indexing of video, in: Proceedings of the International Conference on Image Processing, Vol. 2, Washington, DC, USA, 1997, pp. 526–529.

[6] J. Meng, S.-F. Chang, CVEPS—A compressed video editing and parsing system, in: Proceedings of ACM Multimedia, Boston, MA, USA, 1996, pp. 43–53.

[7] V. Kobla, D. Doermann, K.-I. Lin, C. Faloutsos, Compressed domain video indexing techniques using DCT and motion vector information in MPEG video, in: Proceedings of the SPIE Conference on Storage and Retrieval for Image and Video Databases V, Vol. 3022, San Jose, CA, USA, 1997, pp. 200–211.

[8] A. Hampapur, R. Bolle, Feature based indexing for media tracking, in: Proceedings of the IEEE International Conference on Multimedia and Expo, Vol. 3, New York, NY, USA, 2000, pp. 1709–1712.

[9] R. Mohan, Video sequence matching, in: Proceedings of the International Conference on Acoustics, Speech and Signal Processing, Vol. 6, Seattle, WA, USA, 1998, pp. 3697–3700.

[10] A.K. Jain, A. Vailaya, W. Xiong, Query by video clip, in: Proceedings of the International Conference on Pattern Recognition, Brisbane, Australia, 1998, pp. 909–911.

[11] R. Lienhart, C. Kuhmunch, W. Effelsberg, On the detection and recognition of television commercials, in: Proceedings of the IEEE Conference on Multimedia Computing and Systems, Ottawa, OT, Canada, 1997, pp. 509–516.

[12] J.M. Sanchez, X. Binefa, J. Vitria, P. Radeva, Local color analysis for scene break detection applied to TV commercials recognition, in: Proceedings of the Visual 99, Amsterdam, Holland, 1999, pp. 237–244.

[13] D.A. Adjeroh, M.C. Lee, I. King, A distance measure for video sequence similarity matching, in: Proceedings of the International Workshop on Multi-Media Database Management Systems, Dayton, OH, USA, 1998, pp. 72–79.

[14] D.M. Squire, H. Muller, W. Muller, Improving response time by search pruning in content based image retrieval system, using inverted file techniques, in: Proceedings of the IEEE Workshop on Content Based Image and Video Libraries, Fort Collins, CL, USA, 1999, pp. 45–49.

[15] G.A. Stephen, String Searching Algorithms, World Scientific, Singapore, 1994.

[16] T.H. Merrett, H. Shang, X. Zhao, Database structures, based on tries, for text, spatial, and general data, in: Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications, Kyoto, Japan, 1996, pp. 507–515.

[17] H. Wang, C.-S. Perng, W. Fan, S. Park, P.S. Yu, Indexing weighted sequences in large databases, in: Proceedings of the IEEE International Conference on Data Engineering, Bangalore, India, 2003, pp. 63–74.

[18] S. Park, W.W. Chu, J. Yoon, C. Hsu, Fast retrieval of similar subsequences of different lengths in sequence databases, in: Proceedings of the IEEE International Conference on Data Engineering, San Diego, CA, USA, 2000, pp. 23–32.

[19] A. Bovik, Handbook of Image and Video Processing, Academic Press, New York, 2000.

[20] A. Hampapur, K. Hyun, R. Bolle, Comparison of sequence matching techniques for video copy detection, in: Proceedings of the SPIE Conference on Storage and Retrieval for Media Databases, San Jose, CA, USA, 2002.

[21] M. Ioka, M. Kurokawa, A method for retrieving sequences of images on the basis of motion analysis, in: Proceedings of the SPIE Conference on Image Storage and Retrieval Systems, San Jose, CA, USA, 1992, pp. 35–46.

[22] D.N. Bhat, S.K. Nayar, Ordinal measures for image correspondence, IEEE Trans. Pattern Anal. Mach. Intell. 20 (4) (1998) 415–423.

[23] H. Shang, Trie methods for text and spatial data on secondary storage, Ph.D. Dissertation, McGill University, 2001.