Mining Recurring Events Through Forest Growing

Junsong Yuan, *Student Member, IEEE*, Jingjing Meng, Ying Wu, *Senior Member, IEEE*, and Jiebo Luo, *Senior Member, IEEE*

***Abstract*—Recurring events are short temporal patterns that consist of multiple instances in the target database. Without any *a priori* knowledge of the recurring events, in terms of their lengths, temporal locations, the total number of such events, and possible variations, it is a challenging problem to discover them because of the enormous computational cost involved in analyzing huge databases and the difficulty in accommodating all the possible variations without even knowing the target.**

We translate the recurring event mining problem into finding temporally continuous paths in a matching-trellis. A novel algorithm that simulates a "forest-growing" procedure in the matching-trellis is proposed. Each tree branch in the resulting forest naturally corresponds to a discovered repetition, with temporal and content variations tolerated. By using locality sensitive hashing (LSH) to find best matches efficiently, the overall complexity of our algorithm is only sub-quadratic to the size of the database. Experimental results on the TRECVID video data of 10.5 hours and a human dance video dataset of 32,260 frames show that our method can effectively and efficiently discover recurring events such as TV commercials from news videos and typical dance moves from human dance sequences, in spite of large temporal and content variations.

***Index Terms*—Event mining, motion pattern discovery, temporal pattern discovery.**

I. Introduction

IF AN EVENT occurs repetitively, it can be a pattern of great interest. The recurrence can be exact repetitions, like commercials in TV programs [1] and popular music in audio broadcasting [2], [3]. The recurrence can also be inexact repeats which are similar to each other and share the same spatial-temporal pattern, for example, the same human actions performed by different subjects as shown in Fig. 1. In video analysis, it is important to automatically discover recurring video events in understanding, organizing, and searching based on video contents. There are many related applications reported in the literature, such as commercial detection and analysis [1], [4], news topic threading and tracking [5], [6], news broadcast structure analysis [7], [8], [9], and many others mentioned in [10].

Manuscript received March 06, 2008; revised July 21, 2008. First published September 26, 2008; current version published October 29, 2008. This work was supported in part by National Science Foundation Grants IIS-0347877 and IIS-0308222. This paper was recommended by Associate Editor S. Maybank.

J. Yuan and Y. Wu are with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208 USA (e-mail: j-yuan@u.northwestern.edu, yingwu@ece.northwestern.edu).

J. Meng is with Motorola Applied Research and Technology Center, Schaumburg, IL 60196 USA (e-mail: jmeng@motorola.com)

J. Luo is with Kodak Research Labs, Rochester, NY 14650 USA (e-mail: jiebo.luo@kodak.com.)

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2008.2005616

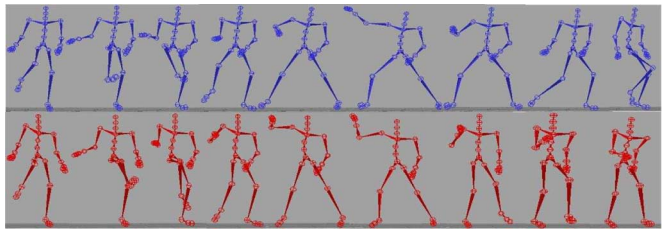


Fig. 1. A typical dance movement in the Michael Jackson-style dance, performed by two different subjects (first and second rows). Such a dynamic motion pattern appears frequently in the Michael Jackson-style dance and is a recurring event in the dance database. The spatial-temporal dynamics in human motions can contain large variations, such as non-uniform temporal scaling and pose differences, depending on the subject's performing speed and style. Thus it brings great challenges in searching and mining them.

Compared with video clip search, where a query clip is usually provided by the user and the task is to find the matches in the video database [5], [11]–[14], the problem of recurring event mining is more challenging, because it has to be unsupervised and blind of a target [3], [15], [16], as these is no query provided. In other words, there is generally no *a priori* knowledge of the events to be discovered, including (i) what the recurring events are; (ii) where they occur in the video; (iii) how long they last; and (iv) how many recurrences there are or even the existence of such a recurring event. Exhaustively searching for the recurrences by checking all possible durations and locations is computationally prohibitive, if not impossible, in large video databases.

Although efficient algorithms have been proposed to reduce the computational complexity in finding exact repetitions [7], [10], mining recurring events remains to be a very challenging problem when the recurring patterns exhibit content or temporal variations (i.e., they are not exact repetitions). For instance, the same video event may vary depending on the encoding scheme and parameters (e.g., frame size/rate, color format), or content changes due to post-editing, not to mention the intra-class variations [17]. Taking the human action patterns as another example, if we treat each typical action as a recurring event, such a recurring pattern can be performed very differently depending on the speed, style, and the subject [18], [19]. As shown in Fig. 1, although the two human actions belong to the same motion pattern, they are far from identical. Consequently, how to handle the possible variations in the recurring events brings extra challenges to the mining problem, especially given the fact that we have no *a priori* knowledge of the recurring pattern [20].

To automatically discover recurring events, our emphasis in this work is not only on exact repeats such as duplicate commercials or music patterns as studied before [7], [10], but also events that are subject to large temporal and spatial variations, such as representative actions in human movements. To this

end, we propose a novel method called "forest-growing" in this paper. First of all, a video or motion sequence is chopped into a sequence of *video primitives* (VPs), each characterized by a feature vector. Suppose the whole database generates in total N VPs, instead of calculating and storing a full $N \times N$ self-similarity matrix as in previous methods [7], [21], [22], for each VP, we query the database and obtain its K best matches. A matching-trellis can be built to store the N query results, which is of limited size $K \times N$ ($K \ll N$). This treatment saves both computational and memory costs, and it is still effective in preserving the best K matches that keep the important information. Based on the matching-trellis, it is clear that a temporally continuous path established in the trellis corresponds to a repetition of a recurring event. Without knowing the location and duration of the repetition, we can grow trees in the matching-trellis, where each branch is a continuous path and is associated with a repetition. The length of the branch can be automatically adapted to tolerate the content and temporal variations. Since the total number of recurring events is unknown, multiple trees need to be grown simultaneously. This process gives the name of our method "forest-growing".

In the proposed method, several techniques are used to make the mining process computationally efficient. To speed up the process of building the trellis, we utilize locality sensitive hashing (LSH) [23] for approximate nearest neighbor (NN) query. LSH is a sub-linear method compared with exhaustive linear search. Considering the fact that we have in total N queries, the actual total complexity of building the matching-trellis is sub-quadratic with respect to the size of the dataset N. Therefore, a large computational gain is achieved. To handle the possible variations in mining recurring events, as well as the potential inaccuracy caused by the approximate nearest neighbor search, a branching factor is introduced in our method in growing continuous branches in the forest. Using a carefully designed message-passing scheme that needs one auxiliary array of size N, we achieve a $O(NK)$ complexity in finding all continuous paths. Thus the overall complexity of our method remains sub-quadratic with respect to the database size, and is memory-friendly compared with the methods that need to store the full $N \times N$ self-similarity matrix.

We highlight the advantages of our forest-growing method as follows:

- it can automatically discover all the recurring events without *a prior* knowledge, and the duration of each recurring event is dynamically determined without exhaustively searching for all possible ones;
- it can handle non-uniform temporal scaling and content variations by using a branching factor in growing the forest;
- it is computationally efficient with an overall sub-quadratic complexity with respect to the dataset size, by using locality sensitive hashing (LSH) and a carefully designed forest-growing process;
- it does not require video shot segmentation and can be easily extended to other types of time-series data.

The remaining of this paper is organized as follows. We briefly describe the related work in the literature in Section II, followed by the description of the proposed forest-growing algorithm in Section III. To test the efficiency of our method, we apply our algorithm to find repetitive commercials in TRECVID news video [24] and report the results in Section V. To demonstrate the effectiveness of our method in handling large spatio-temporal variations, we run the algorithm on a 32,260 frame motion captured human dance data to discover recurring motion patterns and report the results in Section VI. We conclude in Section VII.

II. Related Work

There have been many works in mining exact repeats from video [7] and audio [3], [10], [2] streams. In [3], an effective on-line audio stream mining system is proposed to extract repetitive audio segments in real time without human intervention. The method depends on robust audio fingerprints and its similarity search is accelerated by taking advantage of the dense sampling rate in audio signals. The boundaries of the repeat segments can be accurately determined by performing an exhaustive search. In [7], a video repetition mining method is proposed, which can discover very short repeats from news videos. These short repeats are program lead-in/lead-out clips that indicate the starting or ending points of a particular TV program. Hence locating these short flag clips can help reveal and understand the structures of the news videos. To speed up the similarity matching process, locality sensitive hashing is applied in [7]. Besides mining repetitive video segments, there are also existing works in finding repeats at the image or video shot level, such as near-duplicate image detection [5] and identical shot detection [6], [25]. However, these methods cannot be directly applied to recurring event mining, where an event can be of arbitrary length and may contain a number of shots.

Other than repetition discovery from videos, there are increasing interests of mining recurring patterns from human motion data as well in the computer graphics literature [26]. As more motion databases become available and their sizes increase, manually labeling and categorizing motions becomes a very time-consuming, if not impossible, task. On the other hand, representative and recurring motion patterns (motion motifs [20]) in human motion data can reveal important semantic structures in human motions, which can be used for motion analysis, automatic database annotation, motion retrieval [27] and motion synthesis from existing data [28], [29]. Due to the large variations in human motions, it greatly challenges the task of mining recurring patterns.

Related works in mining repetitive patterns from music have also been reported in the literature. For example, the key melody that appears repetitively in the music can be used in analyzing the themes of the song. In [30], a music repetition mining method is proposed that can tolerate significant variations in parameters, such as dynamics, timbre, execution of note groups, modulation, articulation, and tempo progression. Discovering such recurring events in music can help understand the music theme and structure [31] and it is helpful to construct indices and facilitate queries for music retrieval applications [32].

Motivated by the successes of mining repetitions from text data, one promising solution of recurring event discovery is to translate temporal sequences (e.g., music, videos, or human motions) to symbolic sequences that are similar to text strings.

Thus, it is hoped that traditional text search and mining methods may be directly applied for temporal events discovery. For example, by treating music as note strings, we can find repetitive music segments by mining common sub-strings [32]. Similarly, by quantizing each video frame into a discrete symbol [8] and translating a video sequence into a DNA-like string, mining recurring events becomes a problem of discovering repetitive motifs from a string database. In spite of successes in previous work [8], [32]–[34], we notice that it is unnecessary to translate multimedia sequences into symbolic strings, in order to discover repetitions. Compared with text data, multimedia data are not characterized as symbolic sequences. For example, in video and motion sequence analysis, a general practice is to characterize a video frame or a human pose as a feature vector. Although mapping the continuous feature vectors to discrete symbols can significantly reduce the dimensionality, it inevitably introduces quantization errors, and it in turn degrades the representation power of the original continuous features, especially in high dimensional feature space. This paper presents a new event mining method that utilizes the continuous video features directly, instead of quantizing them into discrete symbolic labels.

ALGORITHM DESCRIPTION

A. Overview

Without loss of generality, we denote the whole database as a long sequence: $\mathcal{V} = \{\mathbf{S}_i\}_{i=1}^{N}$, where $\mathbf{S}_i$ is the *video primitive* (VP). Depending on the application, $\mathbf{S}$ can be a video segment or an individual frame. After feature extraction, each $\mathbf{S}_i$ is characterized by a d-dimensional feature vector: $\mathbf{S}_i \in \mathbb{R}^d$.

We define that a sub-sequence $\mathcal{V}_1 \subset \mathcal{V}$ belongs to a *recurring event*, if it is similar to another subsequence $\mathcal{V}_2 \subset \mathcal{V}$. In such a case, both $\mathcal{V}_1$ and $\mathcal{V}_2$ are *recurring instances* of the same recurring event $\mathcal{E} = \{\mathcal{V}_1, \mathcal{V}_2\}$. For a recurring event $\mathcal{E}$, it is possible that its multiple instances vary from each other. As a data mining problem, before pattern discovery, it is unclear how long the recurring instances are, where they are, and how many of them there are. The algorithm is required to discover all these repetitions of various lengths and content.

Although it is difficult to discover recurring events of unknown lengths, it is straightforward to discover the repetitions of individual VPs. Given $\mathbf{S}_i \in \mathcal{V}$, we denote all of its best matches as its *matching set*

$$\mathcal{M}_{\mathbf{S}_i} = \left\{ \mathbf{S}_j : \|\mathbf{S}_i - \mathbf{S}_j\| \le \epsilon,\ \forall\, j, |i-j| > \hat{N} \right\} \quad (1)$$

where ϵ is the similarity threshold; $\|\cdot\|$ denotes the dissimilarity measurement, e.g., Euclidean distance; $\hat{N}$ is the minimum temporal distance used to filter similar matches caused by temporal redundancy. Hence $\mathcal{M}_{\mathbf{S}_i}$ does not include the temporal neighbors of $\mathbf{S}_i$.

To make the notation consistent, we assume that the average size of $\mathcal{M}_{\mathbf{S}_i}$ is K and describe the *matching-trellis* $\mathcal{M}_{\mathcal{V}}$ as a $K \times N$ matrix, where each column stores a matching set $\mathcal{M}_{\mathbf{S}_i}$. As briefly explained in Fig. 2, mining recurring events can be translated into the problem of finding continuous paths in the trellis $\mathcal{M}_{\mathcal{V}}$, where each continuous path corresponds to a recurring instance. In the following, we discuss in detail how to efficiently build the matching-trellis in Section III-B, and how to efficiently find the continuous paths in Section III-C. Finally, how to cluster all discovered recurring instances into event groups is discussed in Section III-D.

B. Step 1. Build the Matching-Trellis

As an overhead of our algorithm, we need to find the best matches for each $\mathbf{S}_i \in \mathcal{V}$, in order to build the matching-trellis. Exhaustive search of best matches is of linear complexity, thus is not computationally efficient considering that we have N queries in total. To find the best matches more efficiently, we use LSH [23] to perform approximate ϵ-NN query for each primitive $\mathbf{S}_i \in \mathcal{V}$. Instead of searching for the exact ϵ-NN, LSH searches for the approximate ϵ-NN, and can achieve sub-linear query time. Hence the total cost of building the trellis is reduced to sub-quadratic given N queries.

We briefly explain how LSH works as follows. Essentially, LSH provides a randomized solution for a high-dimensional ϵ-NN query problem. It sacrifices accuracy to gain efficiency. In LSH, there is a pool of hash functions. Each hash function $h(\cdot)$ is a random linear mapping from vector $\mathbf{S}$ to an integer, $h : \mathbb{R}^d \to \mathbb{N}$

$$h_{\mathbf{a},b}(\mathbf{S}) = \left\lfloor \frac{\mathbf{a} \cdot \mathbf{S} + b}{r} \right\rfloor$$

where $\mathbf{a}$ is a random vector of d-dimension and b is a random variable chosen uniformly from $[0, r]$. Under a specific hash function, two vectors $\mathbf{S}_p$ and $\mathbf{S}_q$ are considered a match if their hash values are identical. The closer $\mathbf{S}_p$ and $\mathbf{S}_q$ are in $\mathbb{R}^d$, the more possible that they have the identical hash value, which is guaranteed by the property of (r_1, r_2, p_1, p_2)-sensitive hash function [23]. By pre-building a set of hashing functions for the database, each new query vector $\mathbf{S}_q$ can efficiently retrieve most of its nearest neighbors by only comparing its hash values against those in the database instead of calculating the pair-wise distances in $\mathbb{R}^d$. Thus large computational cost can be saved.

However, despite the large efficiency gain from LSH, as a solution to approximate NN search, LSH may result in problems of missed retrieval. To compensate for the inaccuracy caused by LSH and to handle the content and temporal variations, we introduce a branching factor in Section III-C for forest-growing. Later, we discuss how to determine the parameter ϵ in the NN search and the branching factor B in Section IV.

C. Step 2. Mining Repetitions Through Growing a Forest

As explained before, each temporally continuous path in the matching-trellis indicates a repetition. However, to find all continuous paths through an exhaustive check is not efficient. As seen in Fig. 2, there are in total K^N possible paths of length N in the trellis, not to mention those of lengths shorter than N.

Motivated by the idea of dynamic programming, we introduce an algorithm that simulates a forest-growing procedure to discover all the continuous paths in a matching-trellis. Every node in the $K \times N$ matching-trellis can be a seed and start a new tree in the forest if it satisfies the following growing condition and does not belong to any existing trees.

Definition 1: Tree Growing Condition: In the matching-trellis, a VP $\mathbf{S}_i \in \mathcal{M}_{\mathbf{S}_q}$ can grow if there exists another

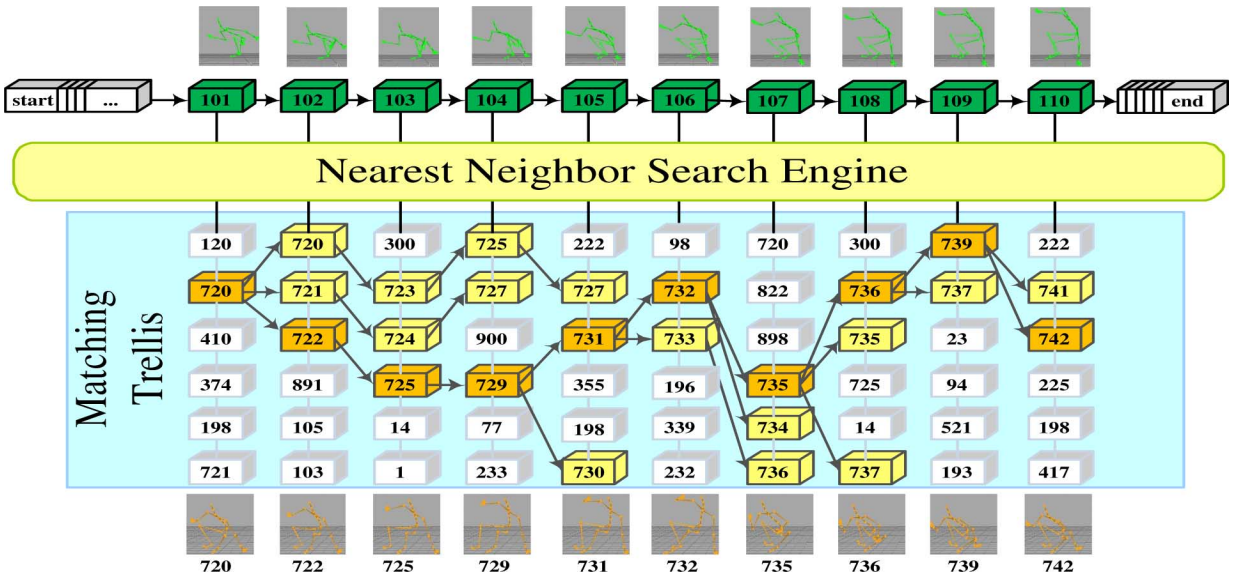


Fig. 2. Mining recurring events through finding continuous paths in the matching-trellis. Each node denotes a primitive $\mathbf{S} \in \mathcal{V}$, labeled by its temporal index. We show part of the whole matching-trellis from column 101 to 110. Given the dataset $\mathcal{V}$, i.e., the top row sequence, we query each $\mathbf{S} \in \mathcal{V}$ and find its K best matches, i.e., each column denotes a matching set $\mathcal{M}_{\mathbf{S}}$. For instance, the matching set of $\mathbf{S}_{101}$ is $\mathcal{M}_{\mathbf{S}_{101}} = \{\mathbf{S}_{120}, \mathbf{S}_{720}, \mathbf{S}_{410}, \mathbf{S}_{374}, \mathbf{S}_{198}, \mathbf{S}_{721}\}$. The highlighted nodes constitute an established tree, which grows from left to right, with the temporal index increasing monotonically from root to leaf nodes. Each tree branch is a continuous path in terms of temporal indices, which indicates a discovered repetition corresponding to the original sequence in the top row. For example, the branch $\{\mathbf{S}_{720}, \mathbf{S}_{722}, \mathbf{S}_{725}, \mathbf{S}_{729}, \mathbf{S}_{731}, \mathbf{S}_{733}, \mathbf{S}_{736}\}$ is a repetition of the top row segment $\{\mathbf{S}_{101}, \mathbf{S}_{102}, \mathbf{S}_{103}, \mathbf{S}_{104}, \mathbf{S}_{105}, \mathbf{S}_{106}, \mathbf{S}_{107}\}$. The longest branch highlighted in orange is picked to represent the whole tree. Although we only show a single tree growing, because the algorithm mimics the process of growing multiple trees simultaneously, we call it "forest-growing".

available $\mathbf{S}_j \in \mathcal{M}_{\mathbf{S}_{q+1}}$, such that $j \in [i, i+B-1]$. Here q, i, j denote the temporal indices, and $B \in \mathbb{N}^+$ is the branching factor that adaptively adjusts the growing speed.

Fig. 2 illustrates one grown tree in the matching-trellis. As a tree grows, it automatically establishes the temporal correspondences between its growing branches and their counterparts in the database (top row segment in dark green in Fig. 2). Repetitions are thus naturally discovered. The total number of repetitive instances is determined by the number of valid trees. The length of a repetition is determined by the length of the longest branch in a tree.

It is worthy noting that the branching factor B plays an important role in the forest-growing procedure. It handles variations in the recurring events and ensures the robustness of our algorithm. Given a tree branch, its temporal index increases monotonically from the root to the leaf node, where the branching factor controls the growing speed. For example, if $B = 3$, a tree branch can grow up to 2 times faster than the original sequence (corresponding top row path as shown in Fig. 2), whose temporal index is strictly increased by 1 in each step. On the other hand, a tree branch can grow much slower than the original sequence when its temporal index increases by 0 in each step. In other words, the growing speed of a branch always adapts to the speed of its corresponding repetition in the top row. Hence we can now accommodate non-uniform temporal scaling among instances of the same recurring event. More importantly, by introducing the branching factor, our algorithm can also tolerate local errors as a tree grows, such as noisy frames or the inaccuracy due to the approximate NN search through LSH. For example, even if LSH fails to retrieve a matching node thus the node does not appear in the next column, the tree still has the chance to grow via the other $B-1$ branches. So the recurring events can still be discovered despite the missed retrieval.

In terms of complexity, since there are in total N columns in the trellis, the algorithm takes $N-1$ steps to finish growing the forest. In each step, we need to check K^2 pairs of nodes between two consecutive columns. Therefore, the total complexity is $O(NK^2)$. To further improve the efficiency, we carefully design a message-passing scheme with one auxiliary index array of size N to speed up each growing step. Each tree branch is described by a message $\{\mathrm{Root}, \mathrm{Length}\}$, where *Root* denotes the temporal index of the tree root and *Length* is the current length of the growing branch. This message is carried by every current leaf node in a tree and will be passed to its descendants as the tree grows. To determine if a leaf node can grow, instead of checking all of the K nodes in the next column in the trellis, we only check the auxiliary array B times to see whether any of its B descendants exists.

Fig. 3 illustrates one growing step from column 312 to 313, using an auxiliary array for speedup. The auxiliary array is essentially a lookup table that tracks the availability of each matching node of 313 and stores the row indices of the matching nodes for tracing back to the trellis. Take one matching node of 312 for example, 927 finds out if it can grow to its descendants (927, 928 and 929) by simply checking the corresponding 3 elements in the auxiliary array. To keep the tree structure, we update the binary flags in the auxiliary array to ensure that each node has only one ancestor. For instance, the binary flags of cells [927–929] are set to 0 after they are taken by node 927 in column 312, so when 312's next matching node 929 grows, it can only branch to node 931 which is still available. In each step, we need to grow K nodes in the current column,

TABLE I
COMPLEXITY ANALYSIS AND COMPARISON. THE COST OF FEATURE EXTRACTION IS NOT CONSIDERED. THE PARAMETER $\alpha > 1$ IS THE APPROXIMATION FACTOR DETERMINED BY ϵ OF ϵ-NN QUERY AND THE CORRECT-RETRIEVAL PROBABILITY p OF LSH. FOR THE METHOD IN [3], L IS A CONSTANT DEPENDING ON THE SAMPLING RATE OF THE VPS

Method	Overhead	Pattern Discovery	Total Complexity	Memory Cost
Naive Exhaustive Search	none	$O(N^3)$	$O(N^3)$	$O(N)$
Self-Similarity Matrix [7]	$O(N^{1+\frac{1}{\alpha}})$	$O(N^2)$	$O(N^2)$	$O(N^2)$
ARGOS [3]	none	$O(N^2/L)$	$O(N^2/L)$	$fixed\ \ size$
Basic Forest-Growing	$O(N^{1+\frac{1}{\alpha}})$	$O(NK^2)$	$O(N^{1+\frac{1}{\alpha}}+NK^2)$	$O(NK)$
Improved Forest-Growing	$O(N^{1+\frac{1}{\alpha}})$	$O(NK)$	$O(N^{1+\frac{1}{\alpha}}+NK)$	$O(NK)$

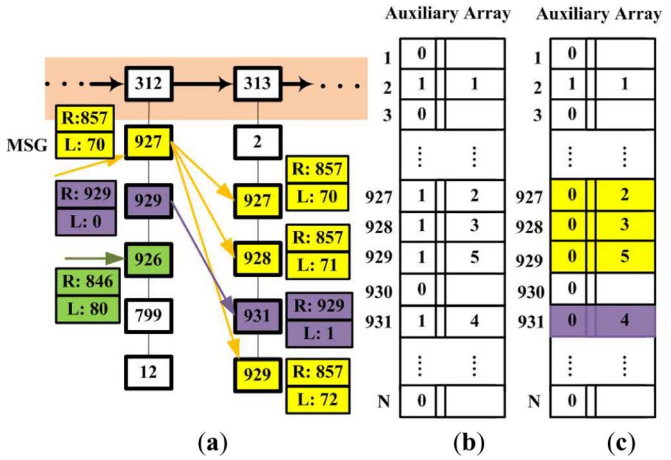


Fig. 3. Improved forest-growing step from column 312 to 313, with branching factor $B = 3$. (a) Two columns 312 and 313 in the matching-trellis; (b) The auxiliary array associated with column 313; each element in the first column stores a binary flag indicating whether the corresponding node is available (0 means not). The second column stores the row index of each node in column 313, e.g., 928 is in the 3rd row of 313; (c) The updated auxiliary array after growing from 312 to 313. In (a), the colored pair of numbers next to each node is the branch message $\{\mathrm{Root}, \mathrm{Length}\}$ to be passed to the descendants during growing, e.g., node 927 belongs to a tree branch whose root is 857 and the current length is 70. When node 927 grows to node 928 in the next column, it updates the message from $\{\mathrm{Root} = 857, \mathrm{Length} = 70\}$ to $\{\mathrm{Root} = 857, \mathrm{Length} = 71\}$ and pass it to 928. The three colors denote different branch status: a live branch (yellow), a new branch (purple) and a dead branch (green). See texts for detailed description.

where each node need to look up the auxiliary array B times. Therefore, the complexity of growing one step in the trellis now becomes $O(KB)$, with a neglectable additional $O(2K)$ cost incurred from clearing and re-initializing the auxiliary array. Given $N-1$ steps in total, the full complexity of the improved forest-growing is now $O(NKB)$, which is more efficient than the previous $O(NK^2)$.

D. Step 3. Clustering Tree Branches

After forest-growing, we only keep the longest branch to represent each tree. The validity of a tree is determined by the length of its longest branch. All valid trees are output to a candidate set $\mathcal{T} = \{T_i : |T_i| \geq \lambda\}_{i=1}^{l}$, where λ is the minimum valid length for pruning invalid trees, and $|T_i|$ denotes the length of the longest tree branch of T_i. Given the candidate set $\mathcal{T}$, we then progressively merge any two trees T_i, T_j with significant temporal overlaps (set to $3/4$ times the length of the shorter branch) to further reduce the redundancy among trees.

After merging highly overlapped trees, we end up with a smaller set of M recurring instances $\mathcal{I}' = \{\mathcal{V}_i\}_{i=1}^{M}$, where each $\mathcal{V}_i$ is described by a message $\{\mathrm{Root}, \mathrm{Length}\}$. To cluster these M instances into G event groups, we measure the similarity between any two instances $\mathcal{V}_i$ and $\mathcal{V}_j \in \mathcal{I}$, again, based on the matching-trellis. Our observation is that if $\mathcal{V}_i$ is similar to $\mathcal{V}_j$, then $\mathcal{V}_j$ should appear in $\mathcal{V}_i$'s matching-trellis. The similarity between two instances is hence defined as

$$s(\mathcal{V}_i, \mathcal{V}_j) = \frac{1}{2}\left[\frac{|\mathrm{sim}(\mathcal{V}_i, \mathcal{V}_j)|}{|\mathcal{V}_j|} + \frac{|\mathrm{sim}(\mathcal{V}_j, \mathcal{V}_i)|}{|\mathcal{V}_i|}\right] \quad (2)$$

where $|\mathrm{sim}(\mathcal{V}_i, \mathcal{V}_j)|$ is the length of the longest branch obtained from growing $\mathcal{V}_j$ in $\mathcal{V}_i$'s matching-trellis. It is notable that $|\mathrm{sim}(\mathcal{V}_i, \mathcal{V}_j)|$ can be different from $|\mathrm{sim}(\mathcal{V}_j, \mathcal{V}_i)|$ as the forest-growing is nonsymmetric.

Finally, based on the resulting $M \times M$ similarity matrix, whose element is $s(\mathcal{V}_i, \mathcal{V}_j)$, we use the normalized cut [35] to cluster these M instances into G groups, where each group corresponds to a recurring event consisting of a number of recurring instances. Besides normalized cut, other advanced clustering methods for time-series data can be applied as well [19], [36].

E. Efficiency and Scalability

A comparison between our algorithm and other methods is summarize in Table I. Both [7] and our method use LSH to accelerate similarity matching, thus have the same overhead cost. However, our pattern discovery procedure through forest-growing in the matching-trellis is more efficient $(O(NK))$ compared with that in the full $N \times N$ matrix $(O(N^2))$. Moreover, our memory cost is lower than [7], since we only store the best K matches in a $K \times N$ trellis, instead of using a $N \times N$ matrix. This presents a great advantage of applying our method to large databases. As an on-line mining method, [3] can perform real-time for broadcast audio streams. Although only linear or fixed memory is required in practice, the worst-case complexity of [3] is still quadratic.

In summary, the proposed forest-growing method has CPU and memory cost comparable with previous methods that focus on mining exact repeats [3], [7], but with added advantages of handling content and temporal variations. Compared with the

basic forest-growing, by using an auxiliary array of length N, the improved version further reduces the complexity of pattern discovery from $O(NK^2)$ to $O(NKB)$, which is essentially $O(NK)$ because B is a small constant as is discussed in Section IV.

III. Discussions of Parameters

A. Branching Factor B

As mentioned before, besides tolerating temporal and local variations, a suitable choice of B can also compensate for the inaccurate matching results caused by LSH. To select a suitable branching factor, we consider the following problem: if there exists a recurring instance of length L in the database $\mathcal{V}$, what is the probability that the instance fails to form a tree branch of length L in the matching-trellis due to the missed retrieval by LSH? Suppose the correct retrieval probability of LSH is p, given branching factor B, the probability of breaking a branch at a given step is

$$\mathrm{Prob}_e = (1-p)^B \tag{3}$$

when all the B descendants are missed by LSH, hence the tree branch cannot grow any longer. Therefore, the probability of breaking a potential tree branch of length L is

$$\begin{aligned}\mathrm{Prob}_t &= 1-(1-\mathrm{Prob}_e)^L \\ &= 1-\left[1-(1-p)^B\right]^L\end{aligned} \tag{4}$$

when any of the L steps breaks.

Hence given p, a large branching factor B decreases the break probability Prob_t, which consequently decreases the probability of missed detection of repetitions. In addition, a potential tree branch can survive large content and temporal variations with a large B.

To investigate how the branching factor B influences the branch length L and its breaking probability Prob_t, we show the relations between L and $Prob_t$ in Fig. 4 with varying B. Here the correct retrieval probability $p = 0.9$ is chosen as the default parameter used for LSH. As can be seen in Fig. 4, if only strict continuous growing is allowed when expanding trees ($B = 1$), the breaking probability Prob_t increases very fast with respect to the branch length L. For example, when $L = 100$, we have $\mathrm{Prob}_t \approx 1$. This means that it is very likely a repetition of length $L = 100$ will be missed due to LSH. In fact, the break probability is already large enough even for short repetitions, e.g., $\mathrm{Prob}_t \approx 0.5$ when $L = 7$. As expected, when more branches are allowed, it is less likely that a continuous branch will break because of missed retrieval. Specifically, when $B = 5$, the breaking probability Prob_t is still small (around 0.1) even for long branches of $L = 10{,}000$.

Although a large B increases the power of handling variations and noises, on the other hand, it may introduce random effects into the tree growing process. An extreme case is when $B = N$, where every possible path in the forest can be a tree branch, which generates meaningless results. In addition, the computational cost of growing the forest ($O(NKB)$) will increase as B increases. In our experiments, we select $B \leq 5$.

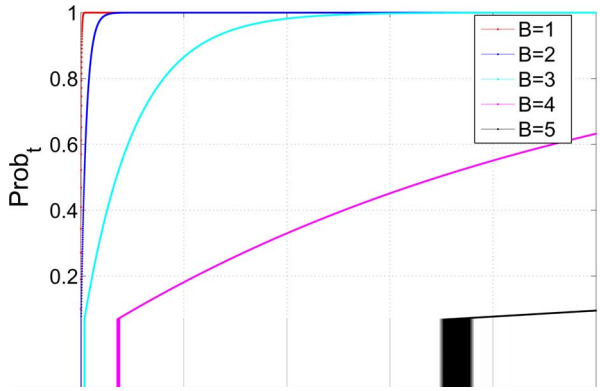


Fig. 4. How the branching factors B influences the relationship between the branch length L and its breaking probability Prob_t. The curve is drawn based on (4), with $p = 0.9$.

B. Nearest-Neighbor Search Parameter ϵ

It is important to select an appropriate ϵ for approximate-NN search as it determines the quantity of the best matches retrieved hence the size of the trellis as well. An improper choice of ϵ will result in either insufficient number of retrieved NNs or an excessive number of NNs [37]. In practice, a small ϵ is preferred for large datasets and memory-constrained conditions, so that the $N \times K$ trellis is of limited size and can be loaded into the main memory easily. On the other hand, a larger ϵ retrieves more NN candidates thus reduces the chance of missed retrieval by LSH.

Considering both requirements, instead of selecting a constant ϵ in (1), we set it as a data-dependent parameter:

$$\epsilon = \mu - \tau \times \sigma \tag{5}$$

where μ and σ are the estimated mean and standard deviation of the pair-wise distance $d(\mathbf{S}_i, \mathbf{S}_j)$, and τ is the parameter controlling the threshold. Under the assumption of a Gaussian distribution, if we select $\tau = 2$, we will retrieve around 2.2% VPs as the NNs. In such a case, we have $K \approx 0.022\,N \ll N$.

C. Minimum Length of Valid Repetition λ

As mentioned in Section III-D, for a discovered repetition to be valid, it must has a minimum length of λ. In the matching-trellis, λ determines the minimum length of valid tree branches. On one hand, to avoid discovery of trivial short repetitions that are caused by noise or randomness, we require λ to be long enough to filter these *pseudo* branches. On the other hand, λ should not be too long so we will not miss valid short repetitions.

To help select appropriate λ, we estimate the probability in forming short repetitions due to the randomness in the matching-trellis. Considering a random matching-trellis of size $N \times K$, where each column, i.e., a matching set, contains K nodes selected randomly from the N candidates. We are interested in the probability of generating a tree branch of length L in this random trellis. The selection of λ should guarantee low probability of finding such repetitions due to random effects.

Each of the $N \times K$ nodes in the trellis has the potential to serve as a root and grow a new tree, if it is not taken by other

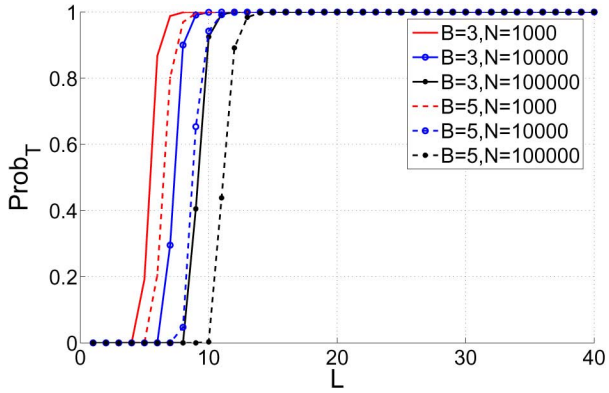


Fig. 5. Selection of effective length λ, with $\epsilon = \mu - 2\sigma$ and $(K/N) \approx 0.022$.

trees yet. If one node can start a new tree, then in the random trellis, the probability that it can grow to the next step is

$$\text{Prob}_g = 1 - \left(1 - \frac{B}{N}\right)^K \tag{6}$$

where $(1 - (B/N))^K$ is the probability that none of K nodes in the next column can be its descendant. Thus the probability for a random tree branch to reach length L (i.e., grows $L-1$ steps) is

$$\text{Prob}_L = \text{Prob}_g^{(L-1)}. \tag{7}$$

In the entire trellis, there are in total $(N-L+1)\times K$ potential nodes that can lead to a tree of length L. Assuming these trees are independent of each other, the probability that *none* of the $(N-L+1)\times K$ nodes can generate a tree of length L is

$$\begin{aligned}\text{Prob}_T &= (1-\text{Prob}_L)^{(N-L+1)\times K} \\ &= \left[1 - \left[1 - \left(1 - \frac{B}{N}\right)^K\right]^{(L-1)}\right]^{(N-L+1)\times K}.\end{aligned} \tag{8}$$

To further investigate the relations between Prob_T and L, we plot in Fig. 5 how $Prob_T$ changes with respect to L, with two sets of pre-specified B and N. It is surprising that Prob_T is sensitive to a fixed threshold of L (e.g., around 10) under given B and N. When L is longer than the threshold, it is highly likely $(\text{Prob}_T \rightarrow 1)$ that a continuous branch of length L will *not* appear in a random matching-trellis, while when L is smaller than the threshold, it is highly likely $(\text{Prob}_T \rightarrow 0)$ that a continuous branch will appear due to random effects. During our forest-growing, we make sure that each discovered tree is of sufficient length $(|T_i| \geq \lambda)$ under the specific parameters N, K and B to rule out those formed at random.

IV. Experiment 1: Recurring Event Discovery From News Video

To evaluate the efficiency of our method in mining exact repetitions, we use broadcast news video as the test set, which contains highly repetitive content including commercials, special program lead-in and lead-out clips, and anchor person shots.

TABLE II
Computational Cost With $\epsilon = \mu - 3\sigma$ and $B = 3$

	Complexity	CPU Cost
build the matching-trellis	$O(N^{1+\frac{1}{\alpha}})$	422.1 sec
forest-growing	$O(NK)$	7.5 sec
total cost	$O(N^{1+\frac{1}{\alpha}} + NK)$	429.6 sec

Considering that commercials are exact repeats and appear frequently, we use them as the ground truth to evaluate the results of our algorithm. The video database contains 22 streams of half-hour ABC news videos collected from the TRECVID dataset [24]. All these half-hour segments are combined into one video sequence with a total length of 10.5 hours. The frame rate is 30 fps and the frame size is 352×240 or 352×264. For evaluation, we collect a set of 56 repetitive clips from the 10.5-hour video dataset as the benchmark data, including 55 commercials and 1 program lead-out clip. The lengths of the commercials vary from 15 to 60 seconds and the program lead-out clip is between 10 and 11 seconds. Each repetitive clip has 2 to 8 recurrences in the database and we manually label the ground truth. These 56 repetitive clips have in total 189 recurring instances in the 10.5-hour dataset.

In our implementation, each video primitive (VP) $\mathbf{S}_i$ is of 4.8 seconds with a sampling interval of 0.4 seconds. Thus each VP has an overlap of 4.4 seconds with its temporal neighbors. In total, the 10.5-hour video database generates $N = 94{,}008$ VPs. We also set the temporal distance $\hat{N} = 300$ VPs to filter neighboring video segments in the matching set. The minimum length of a valid repetition is set to $\lambda = 15$ VPs as suggested in Fig. 5. As a result, a valid repetition is at least $4.8 + (15-1) \times 0.4 = 10.4$ seconds. The experiment is performed on a standard pentium-4 3.19 GHz PC with 1 GB RAM. The algorithm is implemented in C++.

A. Visual Signature Extraction

For exact repeat mining, it is desirable that the visual features are robust under video coding variations, such as compression rate, frame size and color format changes. Moreover, the visual features should also be unique enough to identify different videos. To this end, we follow the feature extraction in [12], [16] and use two types of compact visual signatures to characterize each video segment $\mathbf{S}_i$: (1) color signature and (2) spatial signature. For each $\mathbf{S}_i$, we concatenate its 3 color histograms and 3 spatial pattern histograms into a single normalized histogram $\mathbf{F}$. Since each individual histogram is of 24-dimension, a VP is characterized by a feature vector: $\mathbf{F} \in \mathbb{R}^d$ $(d = 24 \times 6 = 144)$. As all of the signatures can be extracted from the MPEG compressed video data directly [11], they cost fewer CPU time.

B. Efficiency

Table II summarizes the computational costs of (1) building the matching-trellis and (2) growing the forest. Only CPU cost is counted while file I/O cost is not included. Overall, our method proves to be very efficient and can mine a 10.5-hour video in about 7.16 minutes. It is notable that the forest-growing step is extremely efficient and only takes 7.5 seconds to grow all

TABLE III
COMPARISON OF BASIC FOREST-GROWING AND IMPROVED FOREST-GROWING ($K = 24$ AND $N = 94008$)

	Basic ($B = 2$)	Imp. ($B = 2$)	Imp. ($B = 3$)
Complexity	$O(NK^2)$	$O(2NK)$	$O(3NK)$
CPU cost	101.4 sec	5.5 sec	7.5 sec

the continuous paths. Therefore, the major computational cost comes from the overhead of building the matching-trellis, which takes 98.25% of the total CPU time, even with the help of LSH. However, compared with computing a full $N \times N$ similarity matrix, the overhead has already been largely reduced.

To estimate the parameters for ϵ-NN query, we randomly select 1% VPs from the whole dataset. The estimated mean of pair-wise distances is $\mu = 0.269$ and the estimated standard variance is $\sigma = 0.060$. Considering that commercials are exact repeats, we set a strict matching criterion with $\epsilon = \mu - 3\sigma = 0.089$. The ϵ-NN query is fast by applying LSH with a small $\epsilon = 0.089$. In our database of size $N = 94{,}008$, the average CPU time for each ϵ-NN query is only 5 milliseconds. Since the average size of the matching set $\mathcal{M}_S$ is 24, the size of the matching-trellis is approximately $K \times N = 24 \times 94{,}008$, which is a much more compact representation compared with the $N \times N$ self-similarity matrix, as $K/N = 0.000255$.

In Table III, we compare the improved forest-growing method with the basic one. By using an auxiliary array, the forest-growing procedure is largely accelerated. In terms of CPU cost, the improved forest-growing method is around 18 times faster than the basic one.

C. Performance

To evaluate the performance of our forest-growing algorithm, we treat the 56 repetitive clip set as the benchmark set, which corresponds to 189 repetitive instances in total. We evaluate the performance of our method by the recall and precision. The recall score is defined as the percentage of the total 189 instances retrieved. It is measured in terms of the video length, where the percentage is calculated by the number of frames that are correctly discovered by the algorithm. This recall score reflects how many of the 189 instances are finally detected. On the other hand, because we do not have the ground truth of all possible repetitions, which includes not only commercials, but also anchor person shots and still images, it is difficult to provide the accurate precision score. So instead we estimate the precision by randomly picking 20 discovered recurring instances and manually checking whether they are really repetitions. The precision score is the percentage of correct ones from the 20 discovered instances.

We compare different branching factors ($B = 2$ and $B = 3$) in Table IV to see how they influence the performance. As expected, a larger branching factor leads to more trees in the forest, as well as more valid branches. For example, when $B = 2$, we get 2086 trees and 35084 valid branches, whose lengths are longer than the minimum valid length λ. In comparison, when $B = 3$, more trees (2905) and much more valid branches (94017) are returned. The average number of valid branches per tree almost doubled when we allow trees to grow more flexibly

TABLE IV
COMPARISON OF DIFFERENT BRANCHING FACTORS ($\epsilon = \mu - 3\sigma$)

	$B = 2$	$B = 3$
# valid branches	35084	94017
# trees	2086	2905
ave. # valid branches per tree	16.82	32.36
# instances (merged trees)	939	926
recall of benchmark data	55.0%	84.3 %
estimated precision	95.0 %	90.0 %

TABLE V
COMPARISON OF DIFFERENT CRITERIA IN SELECTION ϵ FOR NN-SEARCH (3σ-CRITERION V.S. 2σ-CRITERION). BRANCHING FACTOR $B = 3$

	$\epsilon = \mu - 3\sigma$	$\epsilon = \mu - 2\sigma$
K	24	430
building trellis	422.1 sec	2741.9 sec
forest-growing	7.5 sec	191.0 sec
# valid branches	94017	706462
# trees	2905	18465
# instances (merged trees)	926	3367
recall of benchmark data	84.3%	98.3%
estimated precision	90.0 %	70.0 %

by increasing the branching factor from $B = 2$ to $B = 3$. Moreover, the results of the benchmark data (189 commercial instances) shows that the larger branching factor $B = 3$ also leads to a higher recall compared with $B = 2$ in mining video repetitions. This result validates our theoretical analysis in Fig. 4. On the other hand, the precision drops as the branching factor increases.

We further compare different selections of ϵ for ϵ-NN search, as shown in Table V. A larger ϵ ($\epsilon = \mu - 2\sigma$) for ϵ-NN query results in a better recall. But it also brings a worse precision score than using a stricter criterion $\epsilon = \mu - 3\sigma$. As expected, the computational time is also much longer with larger ϵ, because more NNs are retrieved. Hence a much larger matching-trellis is built and needs to discover.

V. EXPERIMENT 2: MINING RECURRING PATTERNS FROM HUMAN MOTION

To validate that our algorithm can handle more general spatio-temporal variations, we further test our algorithm on motion captured human motion data to discover recurring motion patterns, which exhibit representative spatio-temporal dynamics in human motions. In the case of human dance data, they are typical dance moves that appear repetitively in a certain type of dance. Our test dataset consists of 32,260 frame human motion data from Carnegie Mellon University Graphics Lab mocap database [38]. It includes 15 motion sequences from 5 different motion categories: *break dance*, *acrobatics*, *Indian dance*, *Michael Jackson-style dance* and *salsa*. These motions are captured in different mocap sessions and performed by different subjects. Since the same type of dance can be performed at different speeds and by different people, instances of the same motion event can vary significantly due to the differences in the skeletal/spatial configurations and speeds. Thus compared with mining exact repeats from videos, mining recurring motion patterns is a much more challenging problem.

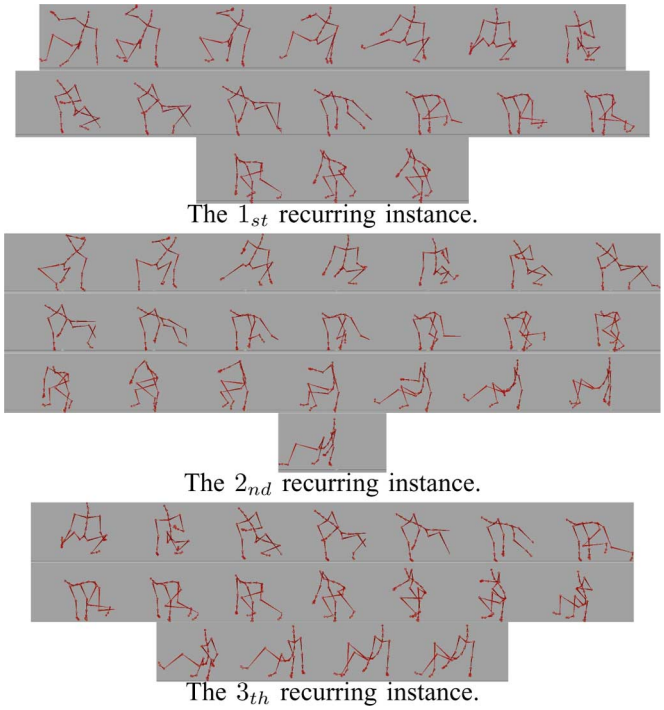


Fig. 6. A discovered recurring event consists of three motion instances. Each instance is a segment in the long motion sequences; the figures are sampled every 10 frames. All of the three instances belong to the same event in break dance, but are of various length and dynamics. More results of the discovered recurring motion patterns can be seen at http://www.ece.northwestern.edu/~jyu410/motionmotifs/.

TABLE VI
DISCOVERY OF RECURRING MOTION PATTERNS. THE ERRORS ARE HIGHLIGHTED. THE TOTAL ERROR RATE FOR CLUSTERING IS 11.29%

	Acrobatics	M.J.	Salsa	Indian	Break	Total
C1	0	11	0	0	0	11
C2	0	0	0	6	0	6
C3	0	0	0	0	3	3
C4	0	0	0	5	0	5
C5	0	0	0	0	9	9
C6	0	0	0	0	4	4
C7	0	0	0	0	2	2
C8	0	3	**1**	0	**2**	6
C9	0	0	0	6	0	6
C10	0	0	0	0	3	3
C11	0	0	0	10	0	10
C12	2	0	0	0	0	2
C13	0	**2**	**2**	5	**1**	10
C14	0	0	0	**1**	8	9
C15	0	0	0	0	16	16
C16	0	0	0	0	3	3
C17	0	0	0	3	0	3
C18	0	0	6	0	0	6
C19	0	0	0	0	6	6
C20	0	0	0	5	0	5
C21	0	0	0	0	4	4
C22	**1**	**1**	**1**	0	10	13
C23	0	0	0	**1**	6	7
C24	0	0	0	0	3	3
C25	0	0	0	0	5	5
C26	0	4	0	0	0	4
C27	0	4	**1**	0	**1**	6
C28	0	0	0	0	3	3
C29	4	0	**4**	0	**1**	9
C30	0	0	**1**	6	0	7
sum	7	25	16	48	90	186

We completely rely on our mining algorithm to discover motion patterns without providing any *a priori* knowledge of the underlying motion data. This experiment is performed on a standard Pentium-4 2 GHz PC with 2 GB RAM.

A. Feature Extraction and Similarity Matching

Given a human motion sequence $\mathcal{V}$, we can represent it as a sequence of human poses $\mathcal{V} = \{\mathbf{S}_i\}_{i=1}^{N}$. Each $\mathbf{S}_i$ is an individual frame and the skeletal pose is characterized by its root position and joint orientations. By treating each pose as a primitive, we measure the similarity between two poses $\mathbf{S}_i$ and $\mathbf{S}_j$ as the weighted Euclidean distance between the quaternion representations of the two poses:

$$d(\mathbf{S}_i, \mathbf{S}_j) = \sum_{k=1}^{J} w_k \left\| q_{i,k} - q_{j,k} \right\| \tag{9}$$

where $\|\cdot\|$ denotes the Euclidean distance; w_k is the weight of joint k; $q_{i,k} \in S^3$ is the unit quaternion representation of the orientation of joint k with respect to its parent joint in frame i; J is the total number of joints in the skeletal representation ($J = 29$ in our test set). As in [21], w_k is set to 1 for important joints like the shoulders, elbows, hips, knees, pelvis, lower back and upper back, whereas w_k is set to 0 for less important joints, like the toes and wrists.

B. Results

Considering the large variations that may exist in human motion data, we choose a less strict 2σ criterion for selecting ϵ, which helps to retrieve similar but not necessarily identical poses given a query. The estimated mean and standard variance of pair-wise distances are $\mu = 3.23$ and $\sigma = 0.94$, respectively, thus $\epsilon = \mu - 2\sigma = 1.35$. The average size of the matching set is $K = 1163$. Branching factor is set to $B = 5$, in order to accommodate the possibly large temporal variations in human motions. When building the match trellis, we filter temporally nearby $\hat{N} = 300$ frames and set the minimum valid length $\lambda = 60$ to eliminate too short motion fragments.

It takes 241 seconds to build a matching-trellis of size 32260×1163, and 213 seconds to find all continuous paths using our forest-growing algorithm. In total, *2,527,496* valid branches are discovered, from which we obtain *6,944* raw paths by only picking the longest branch from each tree. Then we iteratively merge raw paths with temporal overlaps greater than 3/4 and finally obtain 186 recurring instances. The average, maximum and minimum length of the instances are 183, 839 and 60, respectively. These 186 recurring instances are then clustered into 30 event groups using normalized-cut based on the similarity matrix defined in (2). In Fig. 6, we show an event cluster containing three similar human dance sequences.

A person with animation expertise checks the effectiveness of our algorithm by visually identifying if instances in each cluster are indeed similar patterns (temporal-spatial variations in the motions are allowed). Table VI presents the instance distribution in each individual cluster, labeled by the animation expert. For a cluster containing instances from different motion categories, we take the majority as the ground truth and evaluate the

error rate by the percentage of mis-clustered instances in this cluster. The final error rate is defined as the weighted sum of the error rates of all 30 clusters, where the weights are based on the number of recurring instances in each cluster. Judged by the expert, 22 of the 30 event classes are 100% accurate, while 8 out of the 30 events contain instances from different motion categories. The resulting error rate is 11.29%.

Interestingly, in one of the erroneous clusters (C27 in Table VI), where the majority belong to Michael Jackson style dance (4 out of 6), the salsa instance and the break dance instance that are mis-clustered into this event actually share a full-body rotation pattern that is similar to the Michael Jackson dance instances.

VI. Conclusions

Because recurring events may not be exact repeats, it poses extreme challenges to algorithms that try to discover and detect such events automatically. To overcome the large computational cost and to handle content and temporal variations, we translate the recurring pattern mining problem into finding continuous paths in the matching-trellis. The proposed forest-growing algorithm is efficient and of only sub-quadratic complexity with respect to the database size. By introducing a branching factor when growing the forest, it is able to handle content and temporal variations, as well as the misses caused by the approximate search using LSH, which is critical for reducing the computational cost of finding best matches.

The experiment with news video demonstrates the efficiency of our method, which can mine a 10.5 hour video within 8 minutes, upon feature extraction. The other experiment on human dance video validates that our method is capable of handling large content and temporal variations, including pose changes and non-uniform time warping in human motion patterns. Our forest-growing algorithm is applicable to different underlying feature representations and therefore can be easily extended to other types of time-series data for mining recurring patterns.

References

[1] R. Lienhart, C. Kuhmuench, and W. Effelsberg, "On the detection and recognition of television commercials," in *Proc. IEEE Conf. Multimedia Comput. Syst.*, 1997, pp. 509–516.

[2] C. Herley, "Extracting repeats from media streams," in *Proc. IEEE Conf. Acoust., Speech, Signal Process.*, 2004, pp. 913–916.

[3] C. Herley, "Accurate repeat finding and object skipping using fingerprints," in *Proc. ACM Multimedia*, 2005, pp. 656–665.

[4] S. ching, S. Cheung, and T. P. Nguyen, "Mining arbitrary-length repeated patterns in television broadcast," in *Proc. IEEE Conf. Image Process.*, 2005, pp. 181–184.

[5] D.-Q. Zhang and S.-F. Chang, "Detecting image near-duplicate by stochastic attributed relational graph matching with learning," in *Proc. ACM Multimedia*, 2004, pp. 877–884.

[6] K. M. Pua, J. M. Gauch, S. E. Gauch, and J. Z. Miadowicz, "Real time repeated video sequence identification," *Comput. Vis. Image Understand.*, vol. 93, no. 3, pp. 310–327, 2004.

[7] X. Yang, Q. Tian, and P. Xue, "Efficient short video repeat identification with application on news video structure analysis," *IEEE Trans. Multimedia*, vol. 9, no. 3, pp. 600–609, 2007.

[8] P. Wang, Z.-Q. Liu, and S.-Q. Yang, "A probabilistic template-based approach to discovering repetitive patterns in broadcast videos," in *Proc. ACM Multimedia*, 2005, pp. 407–410.

[9] X. Naturel and P. Gros, "Detecting repeats for video structuring," *Multimedia Tools Applicat.*, vol. 38, pp. 233–252, 2008.

[10] C. Herley, "Argos: Automatically extracting repeating objects from multimedia streams," *IEEE Trans. Multimedia*, vol. 8, no. 1, pp. 115–129, 2006.

[11] J. Yuan, Q. Tian, and S. Ranganath, "Fast and robust search method for short video clips from large video collection," in *Proc. IEEE Conf. Pattern Recognit.*, 2004, pp. 866–869.

[12] J. Yuan, L.-Y. Duan, Q. Tian, and C. Xu, "Fast and robust short video clip search using an index structure," in *Proc. ACM Multimedia Workshop on Multimedia Inf. Retrieval*, 2004, pp. 61–68.

[13] A. Joly, O. Buisson, and C. Frlicot, "Content-based copy detection using distortion-based probabilistic similarity search," *IEEE Trans. Multimedia*, vol. 9, no. 2, pp. 293–306, 2007.

[14] C.-Y. Chiu, C.-S. Chen, and L.-F. Chien, "A framework for handling spatiotemporal variations in video copy detection," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 3, pp. 412–417, 2008.

[15] L. Xie, H. Sundaram, and M. Campbell, "Event mining in multimedia streams," *Proc. IEEE*, vol. 96, no. 4, pp. 623–647, 2008.

[16] J. Yuan, W. Wang, J. Meng, Y. Wu, and D. Li, "Mining repetitive clips through finding continuous paths," in *Proc. ACM Multimedia*, 2007, pp. 289–292.

[17] D. Xu and S.-F. Chang, "Visual event recognition in news video using kernel methods with multi-level temporal alignment," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2007, pp. 1–8.

[18] W. Hu, X. Xiao, Z. Fu, D. Xie, T. Tan, and S. Maybank, "A system for learning statistical motion patterns," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 9, pp. 1450–1464, 2006.

[19] T. Xiang and S. Gong, "Abnormal behaviour recognition through incremental learning," *Comput. Vis. Image Understand.*, vol. 111, no. 1, pp. 59–73, 2008.

[20] J. Meng, J. Yuan, M. Hans, and Y. Wu, "Mining motifs from human motion," in *Proc. EUROGRAPHICS*, 2008, pp. 255–258.

[21] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," *ACM Trans. Graphics*, vol. 21, pp. 491–500, Jul. 2002.

[22] L. Kovar and M. Gleicher, "Automated extraction and parameterization of motions in large data sets," *ACM Trans. Graphics*, vol. 23, pp. 559–568, Aug. 2004.

[23] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distribution," in *Proc. 20th Annu. Symp. Computat. Geom.*, 2004, pp. 253–262.

[24] Trec Video Retrieval Evaluation. NIST, 2004 [Online]. Available: http://www-nlpir.nist.gov/projects/trecvid/

[25] S. Satoh, "News video analysis based on identical shot detection," in *Proc. IEEE Conf. Multimedia Expo*, 2002, pp. 69–72.

[26] C. Faloutsos, J. Hodgins, and N. Pollard, "Database techniques with motion capture," in *SIGGRAPH '07: ACM SIGGRAPH 2007 Courses 21*, 2007.

[27] M. Muller, T. Roder, and M. Clausen, "Efficient content-based retrieval of motion capture data," *ACM Trans. Graphics*, vol. 24, pp. 677–685, Jul. 2005.

[28] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," *ACM Trans. Graphics*, vol. 21, pp. 473–482, Jul. 2002.

[29] O. Arikan, D. A. Forsyth, and J. F. O'Brien, "Motion synthesis from annotations," *ACM Trans. Graphics*, vol. 22, pp. 402–408, Jul. 2003.

[30] M. Mller and F. Kurth, "Towards structural analysis of audio recordings in the presence of musical variations," *EURASIP J. Appl. Signal Process.*, vol. 1, no. 2, pp. 200–208, 2007.

[31] L. Lu, M. Wang, and H.-J. Zhang, "Repeating pattern discovery and structure analysis from acoustic music data," in *Proc. ACM Multimedia Workshop on Multimedia Inf. Retrieval*, 2004, pp. 275–282.

[32] J.-L. Hsu, C.-C. Liu, and A. L. P. Chen, "Discovering nontrivial repeating patterns in music data," *IEEE Trans. Multimedia*, vol. 3, no. 3, pp. 311–325, 2001.

[33] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan, "Detecting motifs under uniform scaling," in *Proc. ACM SIGKDD*, 2007, pp. 844–853.

[34] Y.-L. Lo, W.-L. Lee, and L. h. Chang, "True suffix tree approach for discovering non-trivial repeating patterns in a music object," *Multimedia Tools Applicat.*, vol. 37, no. 2, pp. 169–187, 2008.

[35] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, 2000.

[36] T. Xiang and S. Gong, "Video behaviour profiling for anomaly detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 5, pp. 893–908, 2008.

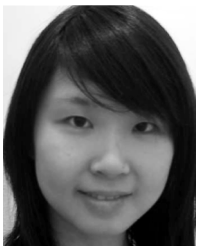
[37] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is nearest neighbor meaningful," in *Proc. Int. Conf. Database Theory (ICDT)*, 1999, pp. 217–235.

[38] CMU Graphics Lab Motion Capture Database. [Online]. Available: http://mocap.cs.cmu.edu/

Junsong Yuan (S'06) is currently a Ph.D. candidate in electrical engineering and computer science at Northwestern University, Evanston, IL. He received the M.Eng. degree from the National University of Singapore in 2005. He was enrolled in the Special Program for the Gifted Young of Huazhong University of Science and Technology, Wuhan, P. R. China, and received the B.S. degree in communication engineering in 2002. His research interests include computer vision, multimedia data mining, and statistical machine learning.

During the summer of 2008, 2007, and 2006, he was a research intern with the Communication and Collaboration Systems group, Microsoft Research, Redmond, WA, Kodak Research Labs, Rochester, NY, and Motorola Labs, Schaumburg, IL, respectively. From 2003 to 2004, he was a research assistant in the Institute for Infocomm Research in Singapore.

Mr. Yuan was awarded the national outstanding student and the Hu-Chunan fellowship in 2001, by the Ministry of Eduction in P. R. China.



Jingjing Meng received the B.E. degree in electronics and information engineering from Huazhong University of Science and Technology, Wuhan, China, in 2003, and the M.S. degree in computer science from Vanderbilt University, Nashville, TN, in 2006.

She was a research intern at Vanderbilt University Institute of Imaging Science (VUIIS) during summer 2005. Since January 2007, she has been a Senior Research Engineer with Motorola Applied Research and Technology Center, Schaumburg, IL. Her current research interests include computer animation and graphics.

Ying Wu (SM'06) received the B.S. degree from Huazhong University of Science and Technology, Wuhan, China, in 1994, the M.S. degree from Tsinghua University, Beijing, China, in 1997, and the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign (UIUC) in 2001.

From 1997 to 2001, he was a research assistant at the Beckman Institute for Advanced Science and Technology at UIUC. During summer 1999 and 2000, he was a research intern with Microsoft Research, Redmond, Washington. In 2001, he joined the Department of Electrical and Computer Engineering at Northwestern University, Evanston, IL, as an Assistant Professor. He is currently an Associate Professor of electrical engineering and computer science at Northwestern University. His current research interests include computer vision, image and video analysis, pattern recognition, machine learning, multimedia data mining, and human-computer interaction.

Dr. Wu serves as an associate editor for IEEE TRANSACTIONS ON IMAGE PROCESSING, *SPIE Journal of Electronic Imaging*, and *IAPR Journal of Machine Vision and Applications*. He received the Robert T. Chien Award at UIUC in 2001, and the NSF CAREER award in 2003.

Jiebo Luo (SM'99) received the B.S. degree in electrical engineering from the University of Science and Technology of China in 1989, and the Ph.D. degree in electrical engineering from the University of Rochester, Rochester, NY, in 1995.

He is a Senior Principal Scientist with Kodak Research Laboratories, Rochester, NY. His research interests include image processing, pattern recognition, computer vision, computational photography, medical imaging, and multimedia communication. He has authored more than 130 technical papers and holds over 40 granted U.S. patents.

Dr. Luo currently serves on the editorial boards of the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE (TPAMI), the IEEE TRANSACTIONS ON MULTIMEDIA (TMM), *Pattern Recognition* (PR), and the *Journal of Electronic Imaging*. He is a Kodak Distinguished Inventor, a winner of the 2004 Eastman Innovation Award, and a fellow of the SPIE.