Mining Motifs from Human Motion

Jingjing Meng[1], Junsong Yuan[2], Mat Hans[1] and Ying Wu[2]

[1]Applications and Software Research Center, Motorola Labs, Schaumburg, IL, USA
[2]Northwestern University, EECS Department, Evanston, IL, USA

Abstract

Mining frequently occurring temporal motion patterns (motion motifs) is important for understanding, organizing and retrieving motion data. However, without any a priori knowledge of the motifs, such as their lengths, contents, locations and total number, it remains a challenging problem due to the enormous computational cost involved in analyzing huge motion databases. Moreover, since the same motion motif can exhibit different temporal and spatial variations, it prevents directly applying existing data mining methods to motion data. In this paper, we propose an efficient motif discovery method which can handle both spatial and temporal variations of motion data. We translate the motif discovery problem into finding continuous paths in a matching trellis, where each continuous path corresponds to an instance of a motif. A tree-growing method is introduced to search for the continuous paths constrained by a branching factor, and to accommodate intra-pattern variations of motifs. By using locality-sensitive hashing (LSH) to find the approximate matches and build the trellis, the overall complexity of our algorithm is only sub-quadratic to the size of the database, and is of linear memory cost. Experimental results on a data set of $32,260$ *frames show that our method can effectively discover meaningful motion motifs regardless of their spatial and temporal variations.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Animation H.2.8 [Database Management]: Data mining

1. Introduction and related work

Motion motifs are recurring motion patterns that exhibit representative spatio-temporal dynamics, such as typical dance moves and martial art actions (e.g. Tae Kwon Do kicks). Each motif consists of a collection of similar short motion segments with varying speeds and spatial configurations. Finding these frequent patterns has many applications in computer animation, like annotating and organizing motion data into different categories to facilitate motion query [MRC05] and synthesis from existing data [KGP02] [AFO03]. However, as more motion databases become available and their sizes increase, manually labeling motifs from long motion sequences becomes a time-consuming, if not infeasible, task.

Previous work in searching similar motion clips from a long sequence is of quadratic complexity of the database size [LCR*02] [KGP02], thus limiting the ability to handle large motion datasets of thousands of frames. To reduce the computation time, [KG04] divide the data set into different categories and build a separate match web for each category. However, this approach cannot be applied to motif mining due to the lack of *a priori* knowledge. Although there exists many data mining methods for discrete data like texts and gene sequences, they cannot be applied to motion data directly. Specifically, human motions are continuous data in a high dimensional space. Hence the same motion pattern can vary largely when performed by different people or at different speeds. It is difficult to handle all possible variations, especially without *a priori* knowledge of the pattern in our mining problem. Some recent work [YKM*07] try to discover motion data by first translating it into discrete symbols before applying traditional data mining to the quantized motion data. However, the discrete translation introduces noise due to quantization errors, and cannot distinguish slight differences among motions under coarse quantization.

In this paper, we present an efficient method for motion motif discovery, which is robust under spatio-temporal variations. Given a motion sequence of N frames, instead of calculating an $N \times N$ similarity matrix [LCR*02] [KG04] to search similar motion segments, we extract a compact $N \times K$ ($K << N$) trellis which only preserves the best matches of each frame. By finding continuous paths on the trellis, we get similar results to searching the match web [KG04], but with a significant memory saving. We apply locality sensitive hashing (LSH) [AI06] [RSH*05] [YWM*07] to speed up building the trellis through approximate nearest neighbor (NN) query. To handle the possible variations of motion patterns (e.g. different speeds) as well as the inaccuracy brought by approximate NN search, a branching factor is introduced to find continuous paths in the trellis. The overall complexity of our method is sub-quadratic to the database size.

© The Eurographics Association 2008.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

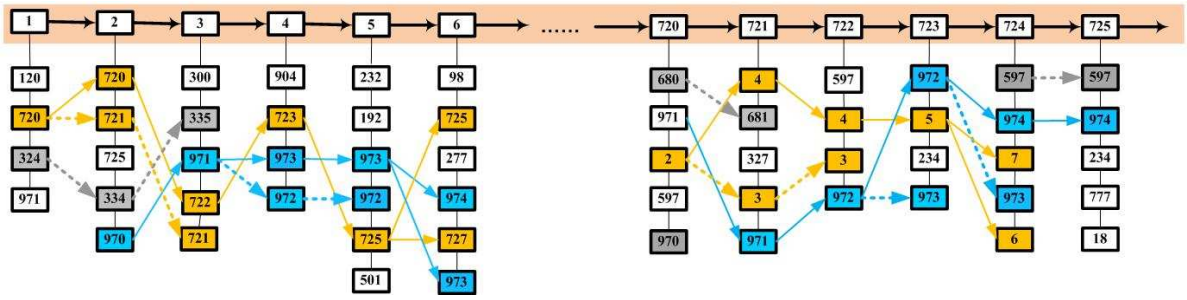

Figure 1: *Matching Trellis: each node denotes a frame in the database, labeled by its temporal index; the top row is a long motion sequence formed by concatenating all motion clips in the database together. The column below a top-row node lists its best matches found in the database. The colored solid paths indicates the continuous paths (in terms of temporal indices) found via our tree-growing algorithm. They are discovered instances of motion motifs and match the motion segment in the top row. Dotted lines indicate invalid paths which are not long enough.*

2. Algorithm description

2.1. Motion motif discovery

Given a motion database, where each frame is a skeletal pose defined by its root position and joint orientations, our goal is to discover frequently occurring motion patterns (i.e. motion motifs). We illustrate our algorithm in Fig. 1, and we explain it in details in the following three steps. In the following, we use N to denote the size of the database (i.e. total number of frames); K is the average number of similar frames found through ε-NN search; and B is the branching factor.

Step 1. Build matching trellis

To find the matching relations among motion frames (poses), we build a compact matching trellis as in Fig. 1. The column below each top-row frame lists its similar matches in the database. The similarity between two frames (poses) f_i and f_j is measured as the weighted Euclidean distance between the quaternion representation of each frame:

$$D(f_i, f_j) = \sum_{k=1}^{J} w_k \left\| q_{i,k} - q_{j,k} \right\|,$$

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 [LCR*02], w_k is set to 1 for important joints like the shoulders, elbows, hips, knees, pelvis, lower back and upper back, whereas w_k for joints like the toes and wrists are set to 0.

It is worth noting that exhaustively calculating pairwise distance to build an $N \times N$ similarity matrix takes $O(N^2)$ [KG04] [LCR*02], which is computational expensive when N is large. In comparison, our trellis is a more compact representation than the similarity matrix since it only keeps the best matches for each frame. To build the matching trellis efficiently, we use LSH [AI06] to perform efficient approximate ε-NN query for each frame. Essentially, LSH provides a randomized solution for the high-dimensional ε-NN query problem. 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 less than $O(N^2)$. We discuss in Section 4 how the inaccurate retrieval results of LSH can be compensated by introducing a branching factor. LSH has been applied to other graphics applications before such as performance animation [RSH*05].

Step 2. Finding continuous paths through tree-growing

Given a matching trellis, finding all the continuous paths is computationally expensive if exhaustively checking all possible paths ($O(K^N)$). Motivated by dynamic programming, [YWM*07] proposed an algorithm that reduces the cost to $O(NK^2)$. Improving upon their method, our algorithm achieves $O(NK)$ by using one auxiliary array of size N to accelerate each growing step.

To grow trees, we start from the 1*st* column in the trellis. A node i can grow if it can find another node $j : j \in [i, i+B-1]$ in the next column, where i, j are the temporal indices, and B is the branching factor that adapts the path growing speed to the top row. As a tree grows, B establishes the temporal correspondences between each growing path and its counterpart in the top row, hence can handle temporal variations. A node can start a new tree if it is not in an existing tree and satisfies the growing condition.

To accelerate tree growing, we use a message-passing scheme, where each path is described by $\{Root, Depth\}$. This message is stored at the current leaf nodes of the growing tree and will be passed to its descendants as the tree grows. Fig. 2 illustrates one growing step of our algorithm. When growing from frame 312 to 313, we first check the matching frames of 313 and update the auxiliary array accordingly. Next, we check each of frame 312's matching frames in the auxiliary array to see if it satisfies the growing condition. Take one matching frame of 312 for example, since 927 finds all its next 3 *temporal* neighbors by simply checking the corresponding 3 cells in the auxiliary array, it generates 3 new branches and passes the path message to the 3 descendants. To keep the tree structure, we ensure that each node only has one ancestor. For instance, the binary value of cell [927-929] are set to 0 after 927 grows, so when 312's next neighbor 929 checks cell [929-931], it can only branch

© The Eurographics Association 2008.

to frame 931 since the binary value of 929 is 0. In each step, we need to query on average K frames and each query cost B binary check. Thus the complexity of one step is $O(KB)$ and the total complexity of tree-growing is $O(NKB)$.

A path dies if it cannot grow any further. We then check its validity by its length, and output valid continuous paths to the candidate path set $P = \{P_i : |P_i| \geq \lambda\}_{i=1}^{T}$, where $\lambda = 60$ frames is the minimum valid length. For instance, in Fig. 2, frame 926 cannot grow any more, since frame 927 and 928 in the next column have both been taken by previous paths. Since the length of the path ending at frame 926 is $926 - 451 = 75$, and the length of the matching path in the top row is 70, both longer than 60 frames, we output two paths: $\{Root = 451, Depth = 75\}$ and $\{Root = 242, Depth = 70\}$.

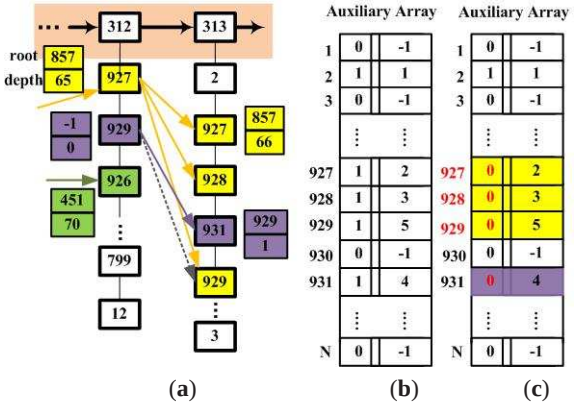

Figure 2: *A tree-growing step from frame 312 to 313. Branching factor $B = 3$. (a) Two column 312 and 313 in the matching trellis; (b) Auxiliary array associated with frame 313; the first column is a binary vector indicating if a frame is matched with 312 (0 means not). The second column contains the row index of each matching frame in the trellis. e.g. 928 is the 3rd neighbor of 313 (c) The same auxiliary array after growing from 312. The colored pair of elements beside each frame is the path information $\{Root, Depth\}$ to be passed to the descendant node while growing. e.g. frame 927 belongs to a path starting from frame 857. It updates $\{Root = 857, Depth = 65\}$ to $\{Root = 857, Depth = 66\}$ and passes the message to its 3 descendant 927, 928 and 929. Three colors denote different path status: live path (yellow), new path (purple) and dead path (green).*

Step 3. Clustering paths

Given the candidate path set P, the final step is to cluster them into motif groups. Since paths obtained from **Step 2** may share the same root(i.e. start frame), we first eliminate this redundancy by sorting paths in P based on their root index, and only keeping the longest path for each tree. Given the resulting longest path set $L = \{L_i\}_{i=1}^{T}$, where T is essentially the number of trees, we then merge any L_i, L_j with significant overlaps (set to $3/4$ times the length of the shorter path). This gives us the set of all motif instances $I = \{I_i\}_{i=1}^{M}$, where $M << N$. To cluster $I_i \in I$ into different motif groups, we measure the similarity between each pair of instances I_i, I_j by the number of similar frames they share. This can be easily done by tracing back to the trellis. Finally, based on the $M \times M$ similarity matrix, we apply normalized cut [SM00] to cluster the M instances into G groups as the final motif sets $C = \{C_i\}_{i=1}^{G}$, where $C_i = \{I_i : I_i \in I\}$. Normalized cut was originally used for image segmentation in computer vision. Here we use it to cluster for motifs.

2.2. Efficiency and scalability

The efficiency of our algorithm with comparison to previous methods is summarized in Table 1. The major cost comes from building the trellis and tree-growing. Comparing with the match web [KG04], which exhaustively computes the distance between every pair of frames in an $N \times N$ matrix, our matching trellis only stores the best matches of each frame, hence reducing the memory cost from $O(N^2)$ to $O(NK)$. The computational cost is also less than their $O(N^2)$ due to the fast ε-NN query using LSH. In addition, we outperform [YWM*07] in path-finding by using an auxiliary array of length N for fast continuation check. Each step, instead of checking the temporal indices of every pair of neighbors of two consecutive frame ($O(K^2)$), we only check the auxiliary array B times for each neighbor, which is ($O(KB)$) with an additional $O(2K)$ operation incurred from clearing and re-initializing the auxiliary array. Hence find continuous paths takes $O(NKB)$ (i.e. $O(NK)$, since B is a small factor) instead of $O(K^2)$.

Table 1: *Computational Cost Comparison*

Method	Trellis Build	Tree-Growing	Total
[KG04]	$O(N^2)$	N/A	$O(N^2)$
[YWM*07]	$O(N^{1+\frac{1}{\alpha}})$*	$O(NK^2)$	$O(N^{1+\frac{1}{\alpha}} + NK^2)$
Ours	$O(N^{1+\frac{1}{\alpha}})$	$O(NK)$	$O(N^{1+\frac{1}{\alpha}} + NK)$

*$\alpha > 1$ is the approximation factor determined by ε and p of LSH.

3. Experimental Results

We tested our method using motion data from Carnegie Mellon University Graphics Lab mocap database. All experiments were ran on a machine with 2G memory and 2GHz processor. To test the effectiveness of our algorithm, we first ran it on a small dataset of 7,690 frames created by concatenating 9 clips together. Three of the nine clips were the same clip (434 frames) manually inserted, which served as the ground truth. Another two sets of similar motion patterns were identified visually by a person with animation expertise. Since human judgment can be inaccurate and subjective, we used them as approximate ground truth and a found motif was considered correct if it overlapped significantly with them. Without merging overlapped clips from tree-growing, our algorithm found 52 motif instances, which were clustered into 8 groups using normalized cut. The 3 manually inserted clips were identified accurately and clustered into one group, along with 5 subsequences. Two other clusters were matching subsequences of the 3 clips. Motion segments similar to the two visually-identified motifs were also found by our algorithm and clustered into another two groups. Interestingly, a new set of motion patterns were discovered as well, which even the animation expert failed to identify. There were one questionable cluster, one wrong

© The Eurographics Association 2008.

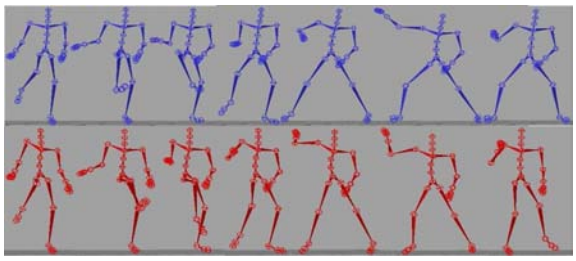

Figure 3: *An example discovered motion motif: a typical Michael Jackson-style dance move. Each row denotes an instance of the motif. Despite the spatio-temporal variations, we correctly discovered these two instances and clustered them into the same motif. See more results in the attached videos.*

cluster, and a cluster containing mostly static frames with similar poses.

To test the scalability of our algorithm, we further ran it on a 32,260 frame dataset consisting of various types of human motions: *break dance*, *acrobatics*, *Indian dance*, *Michael Jackson-style dance* and *salsa*. We totally relied on our mining algorithm to discover the motion motifs without providing any *a priori* knowledge of the data. It took 118 seconds to build a trellis of 32260×1163, and 225 seconds to find the continuous paths. By merging overlapped segments, we obtained 166 motif instances, and clustered them into 50 motifs. The average, maximum and minimum lengths of the motif instances were 206, 835 and 61 respectively. These motifs captured many typical dance moves from different types of dances. Instances within each cluster exhibited various spatial and temporal configurations. Due to the page limit, here we only show one example discovered motifs containing two instances of a typical *Michael Jackson-style* dance move in Fig. 3.

4. Discussions

Branching factor B. By introducing branching factor B, we can accommodate non-uniform temporal scaling by adaptively adjusting the length when growing the path. Also it compensates the inaccuracy brought by mocap noise or the approximate NN search of LSH. Suppose the correct retrieval probability of LSH is p, given branching factor B, the probability of breaking a path at a given step is only $Prob_e = (1-p)^B$, when all the B descendants are miss detected. Therefore the total error probability of breaking a continuous path of length L is $Prob_t = 1-(1-Prob_e)^L$ when any of the L steps breaks. Hence given p, increasing B reduces the probability of breaking a path, but large B increases the computational cost of path growing ($O(NKB)$). In our experiments, we set $B = 5$.

ε in ε-NN search. Since the average length of columns in the trellis (K) is determined by ε, in huge databases, smaller ε is preferred so the $N \times K$ trellis can be loaded into the memory. On the other hand, larger ε reduces the chance of miss retrieval in the ε-NN query by LSH. Considering both requirements, we set ε to $\mu + 2\sigma$ in this paper, where μ and σ are estimated mean and standard variance of the pair-wise distance [BGRS99]. Specifically, for the 32,260 frame data set, $\varepsilon = 1.3$, $\mu = 3.23$, and $\sigma = 0.94$.

5. Conclusion

In this paper, we present a novel method to quickly and automatically find recurring patterns from motion data. By using LSH for fast similarity search and tree growing to find continuous paths, our method is efficient ($O(N^{1+\frac{1}{\alpha}})$) and saves memory cost from $O(N^2)$ to $O(NK)$ compared to [KG04]. Experiments on a 32,260 frame data set shows that our tree growing method based on branching factors is capable of handling spatio-temporal variations. Future work will include applying our motif mining method to motion retrieval, and building motion graphs on complex motion datasets.

Acknowledgment

This work was supported in part by the National Science Foundation Grants IIS-0347877 and IIS-0308222.

References

[AFO03] ARIKAN O., FORSYTH D. A., O'BRIEN J. F.: Motion synthesis from annotations. *ACM Transactions on Graphics 22*, 3 (July 2003), 402–408.

[AI06] ANDONI A., INDYK P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *the Symposium on Foundations of Computer Science (FOCS'06)* (2006).

[BGRS99] BEYER K., GOLDSTEIN J., RAMAKRISHNAN R., SHAFT U.: When is nearest neighbor meaningful? In *Proceedings of the 7th International Conference on Database Theory* (1999), pp. 217–235.

[KG04] KOVAR L., GLEICHER M.: Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 559–568.

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Transactions on Graphics 21*, 3 (July 2002), 473–482.

[LCR*02] LEE J., CHAI J., REITSMA P. S. A., HODGINS J. K., POLLARD N. S.: Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics 21*, 3 (July 2002), 491–500.

[MRC05] MULLER M., RODER T., CLAUSEN M.: Efficient content-based retrieval of motion capture data. *ACM Transactions on Graphics 24*, 3 (July 2005), 677–685.

[RSH*05] REN L., SHAKHNAROVICH G., HODGINS J. K., PFISTER H., VIOLA P.: Learning silhouette features for control of human motion. *ACM Transactions on Graphics 24*, 4 (Oct. 2005), 1303–1331.

[SM00] SHI J., MALIK J.: Normalized cuts and image segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence* (2000).

[YKM*07] YANKOV D., KEOGH E., MEDINA J., CHIU B., ZORDAN V.: Detecting motifs under uniform scaling. In *Proc. SIGKDD '07* (2007), pp. 844–853.

[YWM*07] YUAN J., WANG W., MENG J., WU Y., LI D.: Mining repetitive clips through finding continuous paths. In *Proc. ACM Multimedia '07* (2007), pp. 289–292.

© The Eurographics Association 2008.