Eliminating the Back-Tracking Step in the Longest Common Subsequence (LCSS) Algorithm for Video Sequence Matching

Werner Bailer
JOANNEUM RESEARCH
Institute of Information Systems & Information Management
Steyrergasse 17, 8010 Graz, Austria
werner.bailer@joanneum.at

ABSTRACT

Specific distance measures have been proposed in order to identify video sequences that are very similar over time but not identical (e.g. repeated takes). One such measure is based on the Longest Common Subsequence (LCSS) algorithm, a variant of the string edit distance. After building a matching matrix back-tracking is performed to identify the longest common subsequence. The modification for video sequence matching is that all sufficiently long subsequences that have gaps below a certain threshold need to be found. This increases the effort for back-tracking from linear to quadratic in the average case. In this paper we propose to eliminate the back-tracking step and instead to determine the matching subsequences during matrix creation with only linear effort.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing

General Terms

Algorithms, Performance

1. INTRODUCTION

A collection of video material often has a high degree of redundancy, not only due to the reuse of identical video segments, but also because there are segments that show nearly identical content. Examples are several takes of a scene in rushes material or an event recorded from several very similarly positioned cameras, which is a typical case in news covered by different broadcasters. These video segments do not only share similar static properties (e.g. color distribution in a frame), but are also similar over time (e.g. camera motion, movement of actors and objects). But they are not identical: objects are at slightly different positions and the temporal alignment of the segments may vary, i.e. there may be omissions and insertions.

A measure based on the Longest Common Subsequence algorithm for the distance between two feature sequences A and B of videos has been proposed in [1]. The LCSS algorithm is a variant of the string edit distance, supporting gaps in the match. The authors of [3] apply the LCSS algorithm to matching trajectories in 2D space and introduce the following thresholds: a real number ϵ that defines the matching threshold between the real-valued 2D vector elements of the sequences and an integer δ that defines the maximum offset in the positions to be matched. In order to adapt the LCSS algorithm to matching video segments the following modifications are made. As each element of the sequence is a multidimensional feature vector, a vector $\theta_{sim} = (\epsilon_1, \ldots, \epsilon_K)$ is defined, that contains the matching thresholds for all K features. Similarly a vector $W = (w_1, \ldots, w_K)$ representing the relative weights ($\sum_k w_k = 1$) of the features is introduced. The offset δ introduced in [3] is not relevant for this problem, as the matching subsequences can be anywhere in the parts. Instead the maximum gap size γ of the subsequence is introduced as constraint.

2. THE BACK-TRACKING STEP

In many dynamic programming algorithms, including the original LCSS algorithm (cf. [2]), a $m \times n$ matrix (where m and n denote the lengths of sequences A and B respectively) is built from matching the elements of the input sequences A and B in a first step, taking time $O(mn)$. An example of such a matrix is shown in Figure 1 (adopted from [2]). In the original LCSS algorithm back-tracking of the longest sequence is performed, starting at the lower right corner of the matrix and following a possible sequence to the upper left corner in time $O(m + n)$. In our algorithm such a sequence might have gaps $> \gamma$. In order to satisfy the gap constraint we need to find alternative sequences that might have smaller gaps. Thus we (i) have to follow all equally long paths (i.e. if at any cell in the matrix going up or left yields an equally long result we have to test both) and (ii) we have to test all sufficiently long sequences (i.e. follow all sequences that end in a value $\geq \theta_{len}$ in the bottom row or right column).

The consequence of (i) is that a back-tracking step takes in the worst case $O((m + n)\log(m + n))$, as we have to follow two possible paths at every cell in the matrix. This worst case happens when the distance between two similar stationary sequences is determined and each element of A

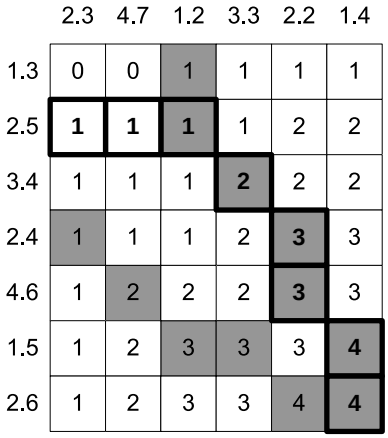

Figure 1: Simplified example of the matching matrix ("c table") of LCSS matching. The values of the one dimensional feature sequences (shown at the top and on the left) are matched with thresholds $\theta_{sim} = 0.5$ and $\theta_{len} = 3$. The LCSS algorithm yields the sequence shown with emphasized border as the longest match. However, if we require a maximum gap $\gamma \leq 1$, this sequence is not a valid result, as there is a gap of two between the first two matching elements. Instead the sequences shown in gray are considered, which are results of the same length that additionally satisfy the gap constraint.

matches each element of B. The consequence of (ii) is that back-tracking has to be done for up to $m + n$ sequences. In practical cases of course not all sequences have a length $\geq \theta_{len}$. However, as $\theta_{len} \ll m \approx n$ for longer sequences, the number of sequences for which back-tracking has to be performed is only reduced by a constant factor. Thus the total effort for the back-tracking step in our algorithm is in the average case $O((m+n)^2)$ and in the worst case $O((m+n)^2 log(m+n))$.

In addition we might get partly overlapping matches that need to be post-processed, adding an additional effort of $O(k^2)$ for the (usually small) number k of matches.

3. BUILDING THE RESULT SEQUENCE DURING MATCHING

For our problem we are not interested in the exact alignment of the two sequences, but only in matching subsequences that have gaps $\leq \gamma$. We can thus eliminate the costly back-tracking step during the construction of the matching matrix as follows. This matrix is built starting in the upper left corner, either line-by-line or column-by-column (depending on whether we match A to B or B to A). We assume without loss of generality (the distance is symmetric) that we are working column-by-column. We keep a list of matching subsequences found so far. For every line i we store the column index of the last match j_{last} found on this line and the matching subsequence this match belongs to.

Every time we encounter a matching element, we check if it continues one of the matching subsequences. If this is the case, the city block distance $L_1(\cdot)$ between the new match at (i, j) and any of the previous matches must be $\leq (\gamma + 2)$. Note that the value 2 comes from the fact that subsequent matches are connected diagonally, which corresponds to a L_1 distance of 2. Thus we find the closest match in the list of matches per line. We only have to check the distances $L_1((i, j), (i_{last}, j_{last}))$ for $\forall i_{last} \in [i-(\gamma+1), i]$, i.e. this check can be done in constant time (with γ being typically small). The new matching element is added to the closest matching subsequence and the list entries are updated. These checks and updates are done for every matching element. In the average case we can assume that each element of A matches none or one from B, yielding $\min(m, n)$ matches and an effort for the checks of $O(\min(m, n))$. In the worst case described above, where each element of A matches every element of B the effort is $O(mn)$.

As the identified matching subsequences could be too short (length $\leq \theta_{len}$) we need to post-process the list of matching subsequences and remove the short ones, requiring an effort of $O(k)$.

4. CONCLUSION

When applying the LCSS algorithm to matching video sequences the maximum length of gaps needs to be constrained. The solution of the algorithm is thus not the single longest common subsequence but all sufficiently long subsequences with gaps shorter then a threshold. This modification increases the effort for the back-tracking step. In this paper we have proposed to eliminate the back-tracking step but find the matching subsequences while creating the matching matrix. This requires $O(\min(m, n))$ in the average case instead of $O((m + n)^2)$ for the back-tracking. In the worst case the effort increases to $O(mn)$, but still less than $O((m+n)^2 log(m + n))$ for back-tracking in the worst case. In addition the effort for post-processing the k matching subsequences is reduced from $O(k^2)$ to $O(k)$.

5. ACKNOWLEDGMENTS

The research leading to this paper has been partially supported by the European Commission under the contracts FP6-027026, "Knowledge space of semantic inference for automatic annotation and retrieval of multimedia content – K-Space" (`http://www.k-space.eu`) and FP6-045032, "Search environments for media – SEMEDIA" (`http://www.semedia.org`).

6. REFERENCES

[1] W. Bailer, F. Lee, and G. Thallinger. Detecting and clustering multiple takes of one scene. In *Proceedings of 14th Multimedia Modeling Conference*, pages 80–89, Kyoto, JP, Jan. 2008.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[3] M. Vlachos, G. Kollios, and D. Gunopoulos. Discovering similar multidimensional trajectories. In *Proc. of the 18th International Conference on Data Engineering*, pages 673–684, San Jose, CA, USA, 2002.