A Search Scheme for a Multi-Dimensional Time Series Database

Miro Enev, Chloé Kiddon, Kathleen Tuite

3/20/2009

Abstract

We propose an indexing and searching scheme for multi-dimensional time series. To avoid the curse of dimensionality we introduce a novel method for structured dimensionality reduction which allows our system to store separate database tables for user-specified sets of dimensions. To find similar matches for a query, we run separate queries on nonintersecting subsets of dimensions of the full query, and then join on the ids of those results to find all candidate matches to the query; then, with fewer candidates, we can apply more costly similarity measures to find the best matches to the query. While checking each table sequentially may take longer than just using one table to index the full time series, the whole system will be faster using a parallelized process. Our system also allows for users to query on a partial set of dimensions efficiently.

We analyzed the effectiveness of our scheme by finding matches to query hand movements in a database. We found that the ability of the user to specify the dimensional groupings of interest adds a high level of flexibility without a significant sacrifice in accuracy with a high average percentage of the top result sets being shared between the output of the full vs. grouped dimensional datasets. The absolute best match to a query was shared in 73 percent of our experiments.

1 Introduction

Many interesting information retrieval problems can be phrased in terms of searching through time series data for contiguous sequences which match a query pattern (within some limit of allowable temporal variation). Although there has been some progress along this line of work, efficient algorithms remained elusive until 2005 when Eamonn Keogh and his colleagues created an integrated time-series matching framework based on dimensionality reduction, indexing, and dynamic time warping.

Keogh's indexing paper defines its functions based on time series of a single dimension. While it is possible to generalize them to multi-dimensional data (sections 4 and 5.1), doing so adds a tremendous amount of complexity and makes the process much slower. Therefore, we developed a system to reduce the dimensionality of the data by splitting up the dimensions into independent groups. For example, for hand movement data where each finger has two sensor's worth of data, it may make sense to split up the full movement's time series by finger. Then, instead of storing and searching through time series of ten dimensions, we only examine two dimensions at a time. We find that this divide and conquer strategy is more efficient than attempting to traverse indexes and apply similarity metrics in the full dimensional space. This process may return different results than if we search through the entire hand space at once; there can be good overall matches that are not returned in the top results for each separate finger. However, we hoped that this reduction in recall would not be substantial.

Our system also allows the user to query the database using a partial set of dimensions. For example, with hand motion data, a user may only care about a movement the index finger is making instead of a full hand motion. Since

we store sets of dimensions independently of each other, our system is more efficient at these types of queries since the full time series in the database will never be examined and only the tables containing the important dimensions will be searched.

2 Related Work

Querying databases for time series data has been discussed in previous literature. Berndt et. al. introduce a technique called dynamic time warping (DTW) for databases, a technique which uses dynamic programming to match two sequences which may have similar structure but vary in time or speed[1]. Using DTW directly, if one wishes to match a query trajectory (a time series), one must scan the entire list of trajectories stored in the database and apply DTW repeatedly, incurring a large computational cost. To improve upon this, an index is required. One approach is to split up the trajectory into N pieces, take the average values of each piece together as a size-N feature vector, and build an index on this feature vector[4]. Chan et. al. are one of several groups to show that Euclidean distance does not work as an index for DTW[5].

Keogh finally hits on exact indexing of dynamic time warping using a special lower bound function that guarantees no false dismissals[2]. This means that trajectories that are sufficiently far away from the query trajectory are not considered. This paper is the basis of our implementation for this project however we extended this work to the multi-dimensional context. The most similar work is by Vlachos et. al. which emphasizes indexing of multi-dimensional time series, including applications to motion capture data[7]. Finally, Keogh et. al. also address motion capture data[3].

The significant difference between these approaches and our proposal lies in the emphasis on dimensionality selection (grouping) for driving the internal query engine structure. We provide client applications the ability to choose subsets of the data's dimensions and based on this specification, our system stores separate tables and indexes for each groups dimensions. In the context of the hand movement dataset, for example, the user can specify groups of joints (i.e. index finger and thumb) that will have an independent R-tree index and table representation for fast searching. The subset of dimensions selected by the user should allow for more selectivity and a considerable speedup for the creation and use of the R-tree index and subsequent DTW comparisons.

Figure 1: Cyberglove with labled joints.

3 Hand Movement Dataset

As a basis for the project, we have obtained a dataset of time-series data of human hand movements, where each movement in the dataset is a measure of the angles of different join sensors over time. The dataset consists of joint angles dowsampled to 20 Hz (from the original 100Hz) from 10 joints (two from each digit) of the left hand recorded using a wireless Cyberglove (Figure 1) worn during 6 hours of self initiated tasks from three subjects for a total of 18 hours of data. The raw dataset allows us access to additional parameters that describe the abduction/adduction of the fingers relative to each other and the thumb relative to the hand; we also have the palm flexion/extension information. For the purposes of this project we chose to focus on the 10 dimensional joint data only. The data was broken down into separate 5 second movement chunks and periods of inactivity were discarded (queries to the database are instances of 10 dimensional 5 second movement

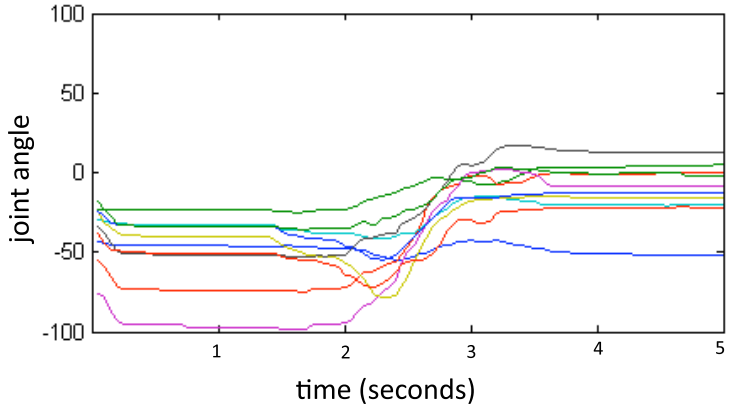

Figure 2: An example 5 second movement chunk - each timeseries represents a single joint's data

chunks). An example movement is illustrated in Figure 2, note that the joint angles can vary between +90 (joint extension) and -90 (joint flexion) degrees.

4 Background Knowledge

The equations in this paper have been taken from [2] and generalized to support multi-dimensional data.

4.1 Dynamic Time Warping

The central means through which we gauge similarity between a query and a member of the database is the dynamic time warping metric (DTW) introduced in 1994 by Berndt and Clifford [1]. Dynamic time warping is an approach using dynamic programming for matching time series data when the overall structure is the same but the time axis might be stretched or compressed (Figure 3). For example, given the continuous, temporal data of the sound of someone speaking, DTW can be used to find instances of the same word pronounced by different individuals despite some vocalization variations.

Suppose we have a query time series $\mathbf{Q}$ of length n with dimensions D and a candidate time series $\mathbf{C}$ of length m also with dimensions D, where

$$\mathbf{Q} = \mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_i}, \ldots, \mathbf{q_n} \quad (1)$$

$$\mathbf{C} = \mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_j}, \ldots, \mathbf{c_m} \quad (2)$$

and

$$\mathbf{q_i} = q_i^1, q_i^2, \ldots, q_i^D$$
$$\mathbf{c_j} = c_j^1, c_j^2, \ldots, c_j^D$$

We can construct an n-by-m matrix E where the (i^{th}, j^{th}) element of the matrix, $e_{(i,j)}$, represents the euclidean distance between the two time points q_i and c_j:

$$e_{(i,j)} = \sqrt{\sum_{d=1}^{D}(q_i^d - c_j^d)^2}$$

We are interested in finding the cost of the best alignment between $\mathbf{Q}$ and $\mathbf{C}$, which is represented by the warping path with the minimal distance between each pair of aligned time points. In general, the warping path may be subject to several constraints [2]. For our purposes, we constrained the path to be monotonically spaced in time and restricted allowable steps in the path to adjacent cells. We do not actually need to know what the warping path is; we are only interested in finding the cost of the minimal alignment. Dynamic Time Warping finds the minimal cost alignment between two time series. At each point the cumulative distance recurrence $\gamma(i, j)$ is found from the distance cost from the current cell's alignment and

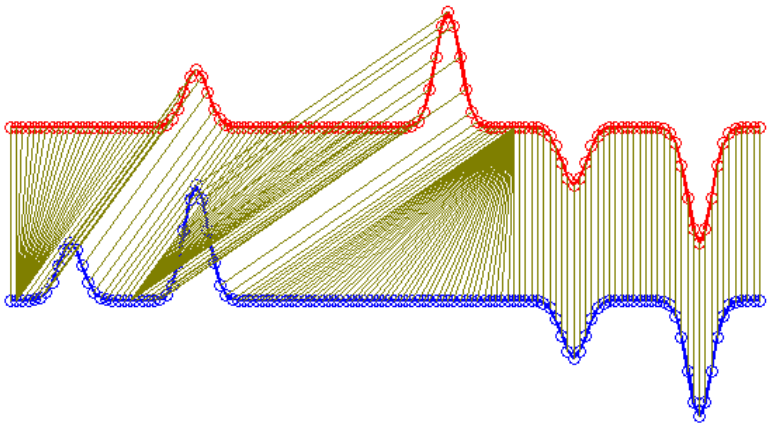

Figure 3: An illustration of the optimal DTW matching between a query (blue) and a candidate (red) timeseries.

the minimum of the cumulative distances from possible previous alignments:

$$\gamma(i,j) = e_(i,j) + min\{\gamma(i-1,j-1), \gamma(i-1,j), \gamma(i,j-1)\} \quad (3)$$

The minimal warping cost returned from the algorithm is then $\gamma(n,m)$.

4.2 R-Trees for Time Series

To store our time series in a database for efficient retrieval we will use a specialized R-tree. R-trees create hierarchical splits in the data space. Each node in the tree stores a *minimum bounding rectangle* (MBR) that spatially bounds the objects indexed in the leaves of that node. MBRs may overlap; therefore, a search for a time series may involve looking through more than one path in the tree.

In general, MBRs are storing static objects in a space. However, we are indexing movements over time. It does not make sense to store the highest and lowest boundaries of the entire movement, since that disregards the contour of the movements over time. Also, the time series may be of variable length so we cannot create just the MBRs to store a boundary for each relative time point. Therefore, for our purposes, we needed to adjust the structure of the minimum bounding rectangles. We assume all time series could be approximated into N segments of equal size[1]. The new MBR structure then stores N bounding rectangles to bound each segment. While this approach with a fixed number of segments will work when indexing time series of variable lengths, it may not be as efficient in this case. For our experiments, we assumed a static size for each time series in the database.

When searching the index for a particular query time series, or inserting a new time series into the index, we calculate piecewise upper and lower bounds for the query. So for a query q with length n, the upper and lower bound series are n-by-D matrices $\mathbf{U}$ and $\mathbf{L}$ defined by:

$$U_{i,d} = max(q^d_{i-r} : q^d_{i+r}) \quad (4)$$

$$L_{i,d} = min(q^d_{i-r} : q^d_{i+r}) \quad (5)$$

where r is the Sakoe-Chiba Band[6]. Then, the piecewise bounding series are defined by N-by-

[1]If the size of the query n is not divisible by N, the last segment will be a little larger to include the leftover time points.

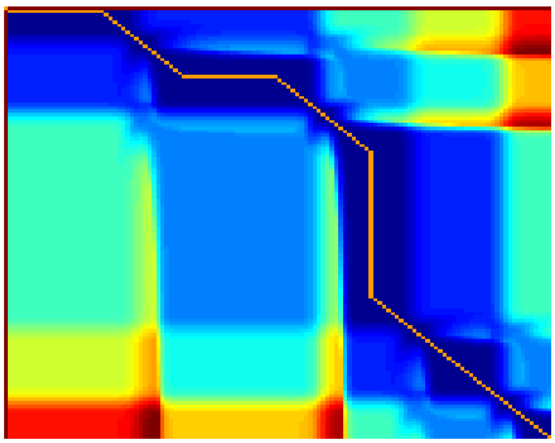

Figure 4: The optimal path through the value matrix used for assigning matches to points in the query and candidate timeseries from Fig 3.

D matrices:

$$\hat{U}_{i,d} = max\left(U_{(\frac{n}{N}i,d)}, \ldots, U_{(\frac{n}{N}(i+1),d)}\right) \quad (6)$$

$$\hat{L}_{i,d} = max\left(L_{(\frac{n}{N}i,d)}, \ldots, L_{(\frac{n}{N}(i+1),d)}\right) \quad (7)$$

5 The System

Our database stores time series split across a set of tables, each with its own R-tree index structure. The R-tree has a specified number of segments N that the data is split into for processing. The inner nodes of the R-tree contain the specialized MBRs as described in section 4.2. The leaf nodes store Piecewise Constant Approximations of the time series they index. For a given time series C of length n with dimensions d, the piecewise approximation is a N-by-d matrix where:

$$\bar{c}_{i,d} = \frac{N}{n} \sum_{j=\frac{n}{N}(i-1)}^{\frac{n}{N}i} c_j^d \quad (8)$$

The owner of the data specifies the dimension splits, if any. In the case of the hand movements, a wise split might be to split the time series data by finger, so that each finger movement gets its own table and index. Full movement data can be recreated by performing a join on all of the joint tables on a key that represents the full movement's id (perhaps a combination of participant id and time taken).

5.1 Search

We used the search algorithm from Keogh 2005[2] to find the K-Nearest Neighbors to a database query. This involves a top-down search through the index R-tree. At each node, the lower-bound distance between the query and each of the node's children is calcuated and added to a priority queue. Finding the optimal matching using DTW requires a costly minimal warping path computation through a distance matrix which would not be feasible to use as a comparison metric in an index of time series (Figure 4). Therefore, lower-bounding metrics are required to simplify the comparison calculations used during indexing while still ensuring that the proper time series in the index are found.

To this end, we utilized two lower-bounding metrics for DTW presented in Keogh and Ratanamahatana 2005: a lower-bounding metric between a time series query and an R-tree node's bounding box, and a lower-bounding metric between a time series query and a

piecewise-approximation of one of the time series in the database. If the node's children are other inner nodes, the MINDIST lower-bounding metric is used. MINDIST measures the distance outside the query's upper and lower bounds that the minimum bounding rectangle falls:

$$MINDIST(\mathbf{Q}, R) = \sqrt{\sum_{i=1}^{N} \frac{n}{N} \sum_{d=1}^{D} \begin{cases} (l_{i,d} - \hat{U}_{i,d})^2 & if\ l_{i,d} > \hat{U}_{i,d} \\ (h_{i,d} - \hat{L}_{i,d})^2 & if\ h_{i,d} < \hat{L}_{i,d} \\ 0 & otherwise \end{cases}} \quad (9)$$

If the node's children are leaves, LB_PAA is used. LB_PAA is a tighter lower bound on the DTW distance between two sequences than MINDIST. It compares the query's upper and lower bounds to a piecewise approximation of the potential match. LB_PAA measures the distance outside the query's upper and lower bounds that the other sequence falls:

$$LB\_PAA(\mathbf{Q}, \bar{C}) = \sqrt{\sum_{i=1}^{N} \frac{n}{N} \sum_{d=1}^{D} \begin{cases} (\bar{c}_{i,d} - \hat{U}_{i,d})^2 & if\ \bar{c}_{i,d} > \hat{U}_{i,d} \\ (\bar{c}_{i,d} - \hat{L}_{i,d})^2 & if\ \bar{c}_{i,d} < \hat{L}_{i,d} \\ 0 & otherwise \end{cases}} \quad (10)$$

The search is directed by the path of least cost. The top of the priority queue is the node searched next. If the top of the queue is a leaf, the full time series indexed by that leaf is retrieved and DTW between the query and the retrieved time series is calculated. The time series id and result is added to a list of potential results. If and when a time series in the list of potential results has a DTW score that is smaller than the lower bound distance of the queue's top element, the time series is added to the set of actual results. The search process continues until K results have been found.

The top K matches from each table are returned and the full time series of each of the total set of matches is found based on id numbers. If the same id number is returned more than once, the full time series for that id is only checked once; therefore, if the number of dimensions has been split among 5 tables, there will be at most 5K full sequences that are considered as possible matches. DTW is performed on each of these potential matches and the K sequences with the best scores are given as the top matches overall.

For queries on partial sets of dimensions, only the tables that contain the specified dimensions are queried. In our present system model, if a user specifies a particular dimension, she must also specify all of the other dimensions that are stored in that same table. For example, for hand movement data where each finger gets its own table, a user cannot query on only one joint sensor; they would have to specify at least both joint sensors for that particular finger. In future work, this constraint may be relaxed.

5.2 Results

5.2.1 Synthetic Experiment

In order to validate the correctness of the methods that underlie our query matching engine, we have created a dataset composed of appended fixed length segments that are randomly mixed. Each segment is one of 8 variants of a Gaussian normal curve. A collection of concatenated segments is a series, and we create multiple independent series to simulate the dimensionality of the joints in the hand. Once we have synthesized the dataset, we can run queries and compare the output of our matching algorithm to the ground truth. Since we can structure the synthetic dataset to have repeating segments amidst the randomly generated data, we imposed guarantees on the nature of the expected results. We successfully tested our solution with the controlled data before moving to the more complex finger joint angle time-series.

5.2.2 Hand Database

To evaluate the performance of our system we ran 2000 randomized queries and recorded the top 9 matches from the full joint dimensions {1,2,3,4,5,6,7,8,9,10} and compared these to the

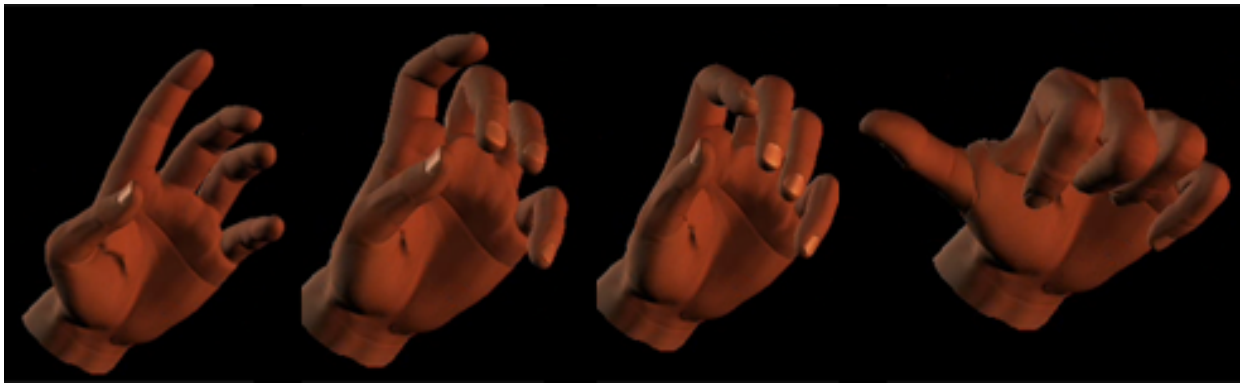

Figure 5: Animation frames (left-to-right ordering) from a highly articulated object manipulation task.

results of the same queries run on groups composed of individual fingers {1,2}, {3,4}, {5,6}, {7,8}, {9,10}. We look at the top 9 non-trivial matches since we return the top 10 results and our query, which is in the database, always matches itself. Based on our analysis, it is not possible to compute traditional precision scores since correct results are returned in all cases. However we can compute recall scores if we consider the overlap between the full joint dimensions results and those of the individuated fingers. We found that an average of 4.395 (out of 9) matches were shared between the full and finger group queries, and that 72.65 percent of the time the top result in the full hand search was retained in the split finger outputs.

On an 8-core Intel Xeon 2.66GHz machine, the full joint test took 3.75 hours to run 2000 queries, and the split joint test took 6.5 hours. This averages out to 6.75 seconds to run one query over one R-tree for the whole hand, and 11.7 seconds per query when split into five R-trees, one per finger. The split query is slower because each R-tree is queried sequentially, when they could be made to run in parallel. We suspect there are other optimizations to our code to speed up our method as well.

To further validate our results we developed a python script that manipulates the parameters of a 3D rendered hand model in Poser Pro to automate the creation of a hand animation with the joint data of a given movement chunk. In most cases the query animations closely resembled the best match animation and it was very interesting to observe the qualitatively different matches that we could produce when we changed the structure of the joint groups that we searched over. A few example frames from a knife twirl are shown in Figure 5.

6 Future Work

One big slowdown in our methods came from the fact that during a search of the R-trees, usually nearly all of the tree was searched to find the best matches. This stems from the fact that R-trees allow nodes' bounding rectangles to overlap. With noisy human hand movement data, they are bound to overlap a lot during a straightforward insertion into the tree. Other R-tree variants exist that ameliorate this problem, such as the R*-tree which uses specialized insertion methods that reduce the overlap of bounding rectangles. This would improve performance during search, but would come at a cost to the set up of the index trees. The tradeoffs need to be investigated further to find the optimal type of R-tree to use.

An additional speedup could be achieved though parallelized processing of the table queries. The user selected groups provide dimensional splits that are natural candidates for parallelization schemes as the processing of the reduced dimensional sets can easily be dis-

tributed amongst shared-nothing cluster nodes which communicate only to convey the results remaining after their indexes have been traversed.

Our work promises to be especially useful to the motor control community which is just beginning to develop intuitions about the structure of the control policies that the brain computes in order to perform complex object manipulations. This database can be leveraged to synthesize learner systems which treat the human as an expert and try to reverse engineer a value function that would inform optimal choice of actions in various tasks.

In conclusion we believe that although our results focused on human hand movements, the methods we propose are general and should be applicable to a range of datasets.

References

[1] Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *AAAI Workshop on Knowledge Discovery in Databases*, 1994.

[2] Eamonn Keogh. Exact indexing of dynamic time warping. In *KIS*, 2005.

[3] Eamonn Keogh, Themistoklis Palpanas, Victor B. Zordan, Dimitrios Gunopulos, and Marc Cardle. Indexing large human-motion databases. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 780–791. VLDB Endowment, 2004.

[4] Eamonn J. Keogh and Michael J. Pazzani. A simple dimensionality reduction technique for fast similarity search in large time series databases. In *PADKK '00: Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications*, pages 122–133, London, UK, 2000. Springer-Verlag.

[5] Franky Kin-Pong Chan, Ada Wai-chee Fu, and Clement Yu. Haar wavelets for efficient similarity search of time-series: With and without time warping. *IEEE Trans. on Knowl. and Data Eng.*, 15(3):686–705, 2003.

[6] Hiroaki Sakoe. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26:43–49, 1978.

[7] Michail Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn Keogh. Indexing multidimensional time-series. *The VLDB Journal*, 15(1):1–20, 2006.