Global Distance-Based Segmentation of Trajectories

Aris Anagnostopoulos † Michail Vlachos ‡ Marios Hadjieleftheriou ⋆ Eamonn Keogh ♣ Philip S. Yu ‡

† Brown University
‡ IBM T.J. Watson Research Center
⋆ AT&T Labs - Research
♣ University of California Riverside

Abstract

This work introduces distance-based criteria for segmentation of object trajectories. Segmentation leads to simplification of the original objects into smaller, less complex primitives that are better suited for storage and retrieval purposes. Previous work on trajectory segmentation attacked the problem locally, segmenting separately each trajectory of the database. Therefore, they did not directly optimize the inter-object separability, which is necessary for mining operations such as searching, clustering, and classification on large databases. In this paper we analyze the trajectory segmentation problem from a global perspective, utilizing data aware distance-based optimization techniques, which optimize pairwise distance estimates hence leading to more efficient object pruning. We first derive exact solutions of the distance-based formulation. Due to the intractable complexity of the exact solution, we present an approximate, greedy solution that exploits forward searching of locally optimal solutions. Since the greedy solution also imposes a prohibitive computational cost, we also put forward more lightweight variance-based segmentation techniques, which intelligently "relax" the pairwise distance only in the areas that affect the least the mining operations.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications, *Data Mining*

General Terms: Algorithms

Keywords: Data simplification, DNA visualization

1. INTRODUCTION

Despite the dramatic increase in processing power, computer systems and networks are still being challenged by the ongoing information avalanche, which necessitates the design of efficient data storage and retrieval mechanisms. Redundant data need to be discarded and potentially useful data can be compressed (simplified), in order to facilitate their efficient visualization, retrieval and processing. In this work, we are attacking the problem of trajectory segmentation, that is, the simplification of a multi-dimensional sequence of values into smaller and simpler primitives, which require a significantly reduced memory footprint than the original object.

Most data-mining tasks operate on a compressed data dimension to speed up operations such as clustering, classification or Nearest-Neighbor search. The data simplification is performed in a disciplined way, so as to provide certain quality guarantees on the data-mining results (such as absence of misclassification). Those guarantees are typically provided by exploiting lower [2] or upper bounds [15] on the distance between the simplified representations. Orthonormal dimensionality reductions techniques (such as SVD, Fourier or wavelets) can provide such guarantees.

Therefore, in this work we provide techniques for "simplification" of trajectories, but in a principled manner. We use Minimum Bounding Rectangles (MBRs) as the simplified primitive structures for approximating trajectories, because of their tight integration with existing multidimensional indexes in commercial DBMS systems (such as R-trees). Additionally, such a representation has been successfully used in a variety of applications, ranging from signature compression of handwritten image data [15], to storage of motion capture data [4, 11] and even for online anomaly detection in time-series data [5]. The notions and techniques presented in this paper are generic enough, and can be adapted into other approximating functions (such as piecewise linear or polynomial functions) with minimal or no changes whatsoever.

Figure 1 illustrates the concept of segmentation on a 2-dimensional trajectory, which is approximated with ten Bounding Rectangles.

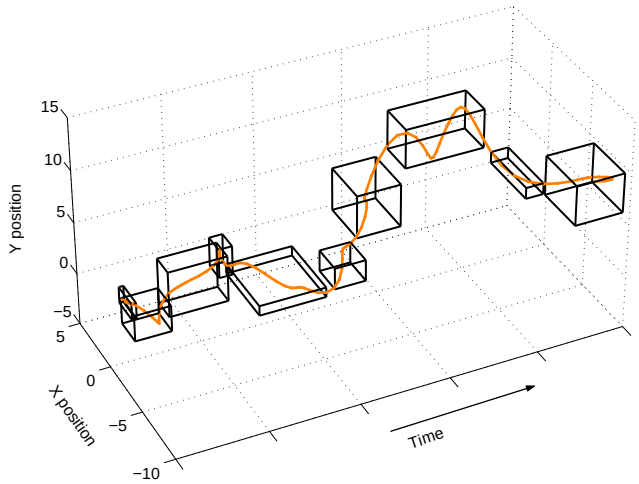

Figure 1: Segmentation of a 2D trajectory into bounding rectangles.

KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.

Trajectory segmentation is important in numerous fields:

1. In spatio-temporal databases the position of moving objects (such as cars on highways) are registered and stored on continually evolving databases. The effective simplification of the object movement can lead to database compaction, thus enabling the fast and efficient support of applications such as matching of trajectory patterns, prediction of future congested areas, and so on.

2. In video tracking, motion segmentation [12] is a common procedure, allowing the effective modeling, characterization and annotation of an object's movement.

3. Sensor network devices have gained momentum lately because of their minuscule size which renders them pervasive in a multitude of applications. Much of the limited energy of these devices is spent on transmitting data, therefore significant savings can be induced by utilizing online versions of data segmentation/ simplification procedures, which can help enhance autonomy of such devices.

4. In many visualization applications the original data are so complex, that cannot be easily manipulated without proper simplification. For example, CAD/CAM systems (e.g., design of an airplane at Boeing) need to facilitate the manipulation and visualization of hundreds of thousands of concurrently visible components [1]. Effective visualization of complex scenes can therefore be achieved only through adaptive object simplification, allowing for faster rendering on screen. Later on, we will depict how the complex DNA structures can be easily visualized and compared using the MBR segmentation techniques.

5. Finally, in video games the approximation of complex objects and their motion trajectories into simplified rectangles is not uncommon, typically for speeding up operations such as collision detection [6].

Previous approaches that approximate trajectories with MBRs, manipulate each trajectory *separately* from the remaining sequences [8], with the objective of minimizing the overlap between MRBs through volume minimization. However, most mining operations are based on distance. Consider, for example, the problem of clustering or outlier detection. With this observation in mind, we try to design segmentation schemes that will approximate as tightly as possible the *inter-trajectory* distances. More accurate distance estimates on the data approximations are bound to provide higher quality mining results.

Therefore, instead of examining separately each trajectory (like previous techniques do), here we present segmentation techniques that collectively utilize information from the whole dataset and attempt to satisfy *global,* distance-based criteria. The contributions and merits of our work are summarized below:

1. We instigate a study of the *global,* distance-based segmentation techniques. We provide a rigorous formulation of the distance-based segmentation problem for trajectories, and show that the exact solution to the problem (based on dynamic programming) is computationally intractable.

2. We present approximate solutions at various computational granularities. The first is a greedy based approximation to the optimal, which nonetheless is still computationally demanding, therefore we also design a variance based technique with significantly reduced running cost and only minimal impact on segmentation quality from a querying perspective.

3. Finally, we demonstrate with an application the usefulness of our techniques for visualization of huge and complex data, such as DNA strings.

2. RELATED WORK

Previous work on trajectory segmentation looked at the problem mainly from a database organization perspective and not from a data-mining point of view. For example [13] examines cost models for evaluating splitting strategies, so as to answer effectively range queries (i.e., "which trajectories pass within area X between times t_1 and t_2"). However, in this work we are interested in optimizing search operations, that is, given a query Q identify as quickly as possible the k most similar trajectories to Q, which reside in the database. The operations that we consider are heavily distance based, which explains our rationale for maximizing the approximated pairwise distances between trajectory segmentations.

Our work also exhibits similarities with the work of [8], which considers local criteria such as volume minimization, for mitigating the effect of MBR overlap in the index organization. To our best knowledge, this is the first work that looks at the problem of trajectory segmentation using global, distance-based criteria.

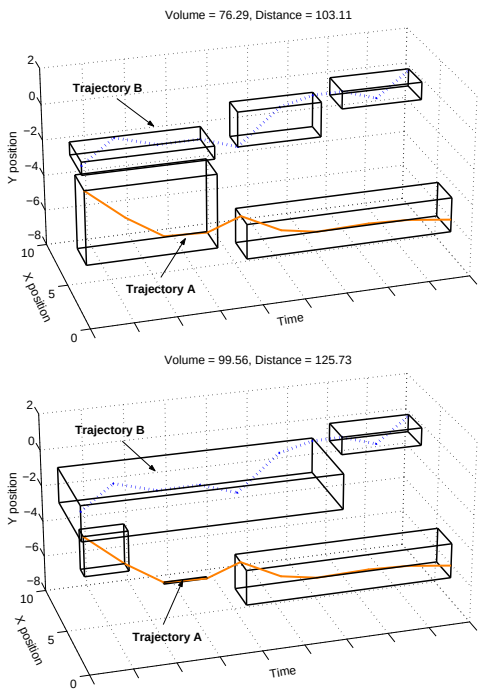

Figure 2: Illustration of distance-based segmentation.

We consider scenarios where the majority of operations involve *comparison* of trajectories (which is the basis of many data-mining operations, such as clustering, aggregation, and so on), therefore, maintaining tight distance approximations

is bound to lead to more accurate results. As already mentioned, previous approaches incorporate different optimization criteria and are not suitable for distance preservation. In order to make this observation more lucid, we depict a brief visual comparison between a volume based segmentation technique and a distance-based one. Figure 2 depicts an example where two trajectories are approximated with 5 MBRs. The top figure illustrates the segmentation achieved by a volume-based optimization criterion, which results in a total volume consumption of 76.29 and pairwise Euclidean distance between the resulting MBRs of 103.11. The bottom schematic illustrates the distance-based segmentation, which (while consuming more space) achieves a tighter distance estimate of 125.73 (i.e., it is closer to the original distance between the raw trajectories).

In the sections that follow we introduce global approximation criteria, based on the total pairwise distances between trajectories. Due to the high computational overhead of the exact approach, in Section 4.2 we present a greedy-based approximation. Section 5 presents an even more lightweight distance-based segmentation technique that utilizes variance for identifying fast candidate areas for "relaxing" the pairwise distance.

3. NOTATION AND PROBLEM FORMULATION

We assume that we are dealing with 2-dimensional trajectories of objects that move on the plane. Extensions to higher-order trajectories are straightforward. We proceed by describing the model and giving some formal definitions.

The input consists of a collection of trajectories $\mathcal{T} = \{T_1, \ldots, T_n\}$. Each trajectory T_i is a mapping $T_i : \{0, 1, \ldots, m-1\} \mapsto \mathcal{R}$, from the set of integers $\{0, 1, \ldots, m-1\}$ (which correspond to discrete time steps) to the range $\mathcal{R}$. In our setting, as we mentioned, usually T_i corresponds to a 2-dimensional trajectory, so $\mathcal{R} = \mathbb{R}^2$, that is, $T_i(t) = (x_i(t), y_i(t))$. Our results hold for any number of dimensions, while we also consider a discrete range set, single dimensional ($\mathcal{R} = \{1, 2, \ldots, R\}$) or multidimensional.

A given trajectory T_i can be approximated by a set of "Minimum Bounding Rectangles" (MBRs) that completely contain the original multi-dimensional sequence on all dimensions (see Figure 1).

For $\mathcal{R} = \mathbb{R}^2$, an MBR B^j for a trajectory T_i is represented as a parallelepiped (rectangle) $B^j = [t_b^j, t_f^j] \times [\ell_x^j, h_x^j] \times [\ell_y^j, h_y^j]$, and we require that $\ell_x^j \le x_i(t) \le h_x^j$ and $\ell_y^j \le y_i(t) \le h_y^j$ for all $T_i(t) = (x_i(t), y_i(t))$ such that $t_b \le t \le t_f$.

Then a segmentation $S(T_i)$ for a trajectory T_i is defined as a set of MBRs, $S(T_i) = \{B^1, B^2, \ldots, B^r\}$, and we require that $t_b^1 = 0$, $t_b^j \le t_f^j$ for $j = 1, \ldots, r$, $t_b^{j+1} = t_f^j + 1$ for $j = 1, \ldots, r-1$, and $t_f^r = m - 1$.

For a segmentation $S(T_i)$ of a trajectory we define $P(S(T_i), t)$ as the projection of the MBR $B^j \in S(T_i)$, for which $t_b^j \le t \le t_f^j$, at time t:

$$P(S(T_i), t) = [\ell_x^j, h_x^j] \times [\ell_y^j, h_y^j],$$

where $t_b^j \le t \le t_f^j$. The area of the projection at time t is

$$\text{Area}(P(S(T_i), t)) = (h_x^j - \ell_x^j)(h_y^j - \ell_y^j).$$

The volume of a segmentation is

$$V(S(T_i)) = \sum_{t=0}^{m-1} \text{Area}(P(S(T_i), t)).$$

The distance between two points $\mathbf{x_i}, \mathbf{x_j} \in \mathbb{R}^2$ is their Euclidean distance:

$$d(\mathbf{x_i}, \mathbf{x_j}) = \|\mathbf{x_i} - \mathbf{x_j}\|_2,$$

but one can consider any other metric. For example, when we consider trajectories that take values in the discrete range $\mathcal{R} = \{1, 2, \ldots, R\}$ we use as distance the function $d(x_i, x_j) = |x_i - x_j|$. We also define the distance between two segmentations at time t as the distance between the rectangles at time t. Formally:

$$d(S(T_i), S(T_j), t) = \min_{\substack{\mathbf{x_i} \in P(S(T_i), t) \\ \mathbf{x_j} \in P(S(T_j), t)}} d(\mathbf{x_i}, \mathbf{x_j}).$$

Finally, the distance between two segmentations is the sum of the distances between them at every time instant:

$$d(S(T_i), S(T_j)) = \sum_{t=0}^{m-1} d(S(T_i), S(T_j), t).$$

The distance between the trajectory MBRs is a lower bound (see Figure 3) of the original distance between the raw data, which is an essential property for guaranteeing correctness of results for most mining tasks (such as kNN search).

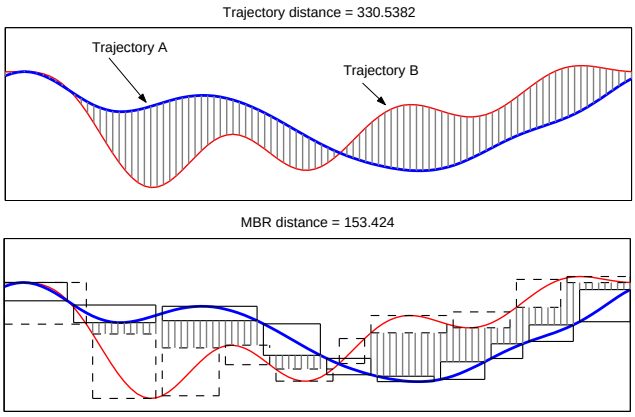

Figure 3: *Top*: Distance between trajectories. *Bottom*: Distance between their respective MBRs (shown in 1D for clarity).

One can define a variety of segmentation problems depending on the function to be optimized. Given a function f that takes as input a segmentation for each trajectory and returns the "cost" of the segmentation (for a minimization problem) the corresponding segmentation problem is posed as follows. Find a segmentation $S(T_i)$ for each trajectory T_i that minimizes

$$f(S(T_1), \ldots, (T_n)).$$

A maximization problem is defined similarly.

Usually the objective is to find segmentations for all the trajectories such that the cost is minimized, given a limited storage capacity (i.e., the total number of rectangles is bounded by some number K).

Typically, the computation of the global minimum (or maximum) of function f is quite costly, and exact approaches might need time and space that increases exponentially to the input size; in Section 4.1 we see some examples. Therefore, one can resort to heuristic approaches that utilize a forward search of locally optimal solutions whose approximation quality in practice is very close to the optimal solution. The skeleton of such an algorithm is given in Figure 4.

1. **Function** forwardSearch($T_1, \ldots, T_n$, K)
2. /* *Assume cost function f_d.* */
3. Start with each point on its own MBR for every trajectory T_i
4. **while** (#MBRs $> K$)
5. Merge the two consecutive MBRs that lead to best local results of f_d
6. Update costs of affected MBRs
7. **end while**

Figure 4: Generic forward search algorithm.

The various greedy segmentation algorithms differentiate themselves at step 2 (optimization function), step 5 (MBR merging) and step 6 (update of statistics). In the next Section we present one such greedy approach where the optimization function is the total distance minimization, while in Section 5 we present another approach where the optimization function is based on variance minimization.

4. DISTANCE-BASED SEGMENTATION

The distance-based segmentation criterion attempts to create MBRs in such a way that the original pairwise distances between all trajectories are preserved as well as possible. Therefore, the objective function is reduced to the following maximization problem:

$$f(S(T_i)_{i=1\ldots n}) = \begin{cases} -\infty, \text{if } \sum_{i=1}^{n} |S(T_i)| > K, \\ \sum_{1\le i<j\le n} d(S(T_i, T_j)), \text{otherwise.} \end{cases} \quad (1)$$

The intuition behind maximizing the pairwise distances is that this would be beneficial for operations such as clustering, or kNN search, since it will provide better inter-object separability.

4.1 A Dynamic Programming Approach

Exhaustive search on the aforementioned problem is computationally prohibitive, as we now show. For simplicity, assume that there are only two trajectories; the extension to multiple trajectories is straightforward. The total number of possible segmentations equals

$$\sum_{K_1=1}^{K-1} \binom{m}{K_1-1}\binom{m}{K-K_1-1} = \binom{2m}{K-2} = \Omega\left(\left(\frac{m}{K}\right)^{K-2}\right).$$

To see why the expression in the left hand side gives the number of segmentations, notice that if the first trajectory contains K_1 rectangles (and each trajectory must have at least one bounding rectangle, so K_1 ranges from 1 to $K-1$), there are $\binom{m}{K_1-1}$ ways to select the starting point for all but the first rectangle, and similarly $\binom{m}{K-K_1-1}$ for the second one. The first equality follows by properties of the binomials. This means that the number of choices that have to be considered grows exponentially with the number of available rectangles K.

A more efficient exact solution is based on dynamic programming. For conciseness we present our main theorem considering only two trajectories, with a range of values taken from a discrete set $\mathcal{R} = \{1, 2, \ldots, R\}$. The results can be extended for multiple trajectories and higher-dimensional spaces, while in the case that the range is continuous we can achieve an approximation by discretizing the range set.

THEOREM 1. *Let T_1 and T_2 be trajectories of length m, taking values in the range $\{1, 2, \ldots, R\}$. There exists a dynamic programming algorithm that assigns K MBRs to two trajectories so that the distance between the resulting segmentations is maximized. The algorithm computes the optimal bounding-rectangle assignment in time $O(KmR^8)$, using space $O(KmR^4)$.*

PROOF. Given in the appendix. □

This algorithm, although polynomial to the range R and the time m, is very impractical even for two trajectories. Even worse, the straightforward extension for more trajectories is exponential in the number of trajectories. Hence, we study approximate solutions to the problem.

4.2 A Greedy Solution

We now begin a study of heuristic approaches to the distance-based segmentation problem, which will allow for a more computationally efficient algorithm, at the expense of the quality of the results. The first algorithm that we present is a greedy forward-search technique, which constitutes a variation on the generic forward-search approach as presented in Figure 4.

For concreteness, assume that we try to solve the problem of maximizing the sum of the distances between the segmentations, as presented in Equation (1). A detailed description of the algorithm appears in Figure 5. For ease of exposition we present a simple variation of the algorithm. In practice, this algorithm can be substantially improved in terms of computational cost, as will be discussed shortly.

The algorithm starts by assigning each point of all trajectories (for a total of nm points) in its own MBR (lines 2–7). Subsequently, as long as the total number of MBRs is larger than K the algorithm merges two consecutive MBRs for any trajectory T_i, such that the merge will result in the least change of the total pairwise distance (lines 11–26). After each merge, the pairwise distances on the affected time instances are updated and the process is repeated (line 10).

THEOREM 2. *Assume that we execute the Forward Search algorithm with n trajectories, m points per trajectory, and K MBRs, where $n \le K \le nm$. Then the time required to execute the algorithm is $O(n^3m^2)$.*

PROOF. First notice that the while loop (lines 10-27) is executed $nm - K$ times. Let us now analyze the running time of each iteration. In every iteration, the algorithm examines all the n trajectories (**for** loop, line 11). Assume that the algorithm examines trajectory T_i and that the corresponding segmentation is $S_i = \{B_i^1, B_i^2, \ldots, B_i^r\}$. In lines 13–22, the algorithm attempts to merge each MBR with the one next to it. If $r = 1$, then there is nothing to do and the algorithm proceeds to the next trajectory. Otherwise it creates the merged MBR (line 14), merging B_i^j with B_i^{j+1}, and calculates the cost difference between the new and the current segmentation (lines 15–16). Line 15 can be executed in constant time, while line 16 can be executed in time $(n-1)(\Delta_j + \Delta_{j+1})$, where $\Delta_j = t_f^j - t_b^j + 1$ is the temporal length of the jth MBR. Since every MBR B_i^j is considered twice—once when it is merged with the previous one and once when it is merged with the next one, except for B_i^1 and B_i^r which are considered only once— the total time to calculate the cost difference for a given segmentation is $(n-1)(2m - \Delta_0 - \Delta_r < 2nm)$. Therefore each iteration of the for loop of line 11 can be executed in time $2n^2m$, so the total running time of the algorithm is bounded by $(nm - K)2n^2m = O(n^3m^2)$. □

1. **Function** forwardSearch($T_1, \ldots, T_n$, K)
2. **for** ($i = 1$ to n)
3. **for** ($j \leftarrow 0$ to $m-1$)
4. $B_i^{j+1} \leftarrow [j,j] \times [x_i(t), x_i(t)] \times [y_i(t), y_i(t)]$
5. **end for**
6. $S_i \leftarrow \{B_i^1, B_i^2, \ldots, B_i^m\}$
7. **end for**
8. #MBRs $\leftarrow n \cdot m$
9. minDist $\leftarrow \infty$
10. **while** (#MBRs $> K$)
11. **for** ($i \leftarrow 1$ to n)
12. /* *assume that* $B_i^j = [t_b^j, t_f^j] \times [\ell_x^j, h_x^j] \times [\ell_y^j, h_y^j]$, *for* $j = 1, 2, \ldots, |S_i|$, *and that* $S_i = \{B_i^1, B_i^2, \ldots, B_i^r\}$ */
13. **for** ($j \leftarrow 1$ to $|S_i| - 1$) /* *estimate the cost of merging* B_i^j *with* B_i^{j+1} */
14. $B_{\text{new}} \leftarrow$ mergeMBRs(B_i^j, B_i^{j+1})
15. $S_{\text{new}} \leftarrow S_i \cup \{B_{\text{new}}\} \setminus \{B_i^j, B_i^{j+1}\}$
16. currentDiff $\leftarrow$ $\sum_{r \neq i} \sum_{t=t_b^j}^{t_f^{j+1}} d(S_i, S_r, t) - d(S_{\text{new}}, S_r, t)$
17. **if** (minDiff $>$ currentDiff) /* *we found a better merging* */
18. minDist $\leftarrow$ currentDiff
19. iBest $\leftarrow i$
20. $S_{\text{best}} \leftarrow S_{\text{new}}$
21. **end if**
22. **end for**
23. **end for**
24. /* *now we perform the best merge that we have found* */
25. $S_{\text{iBest}} \leftarrow S_{\text{best}}$
26. #MBRs $\leftarrow$ #MBRs $- 1$
27. **end while**

1. **Function** mergeMBRs(B_i^j, B_i^{j+1}) /* *returns the MBR that results after combining the two consecutive MBRs* B_i^j *and* B_i^{j+1} */
2. **return** $[t_b^j, t_f^{j+1}] \times [\min\{\ell_x^j, \ell_x^{j+1}\}, \max\{h_x^j, h_x^{j+1}\}] \times [\min\{\ell_y^j, \ell_y^{j+1}\}, \max\{h_y^j, h_y^{j+1}\}]$

Figure 5: Greedy distance-based segmentation algorithm.

Notice that the analysis above considers a straightforward, albeit inefficient implementation; our implementation is more efficient. For example, many quantities are being calculated multiple times. A more efficient implementation can maintain a priority queue with the cost difference of merging two consecutive MBRs belonging to the same trajectory. When the two MBRs at the top of the queue are merged, one can update only the cost differences of the other MBRs that are affected (the ones that intersect with the merged MBRs in time). Nevertheless, even this simplified analysis demonstrates that while the greedy approach achieves a significant computational leap from the exact solution, for practical purposes the usefulness of the greedy algorithm can be quite limited.

Another greedy approach is the backward search: the algorithm initially approximates each trajectory in a single MBR and then starts splitting, until it eventually reaches K MBRs, always performing the split that leads to the least distance deterioration. Nevertheless, even though this approach superficially seems less expensive, in practice the bookkeeping costs per iteration render it even more expensive than the forward search approach.

5. VARIANCE-BASED SEGMENTATION

Even though the greedy approach is much more efficient than the exact solution, it is still prohibitively expensive when we are dealing with very large datasets and lengthy trajectories. Indicatively, some simulations presented in Section 6 required several days to run to completion. Therefore, we attempt to develop even faster algorithms.

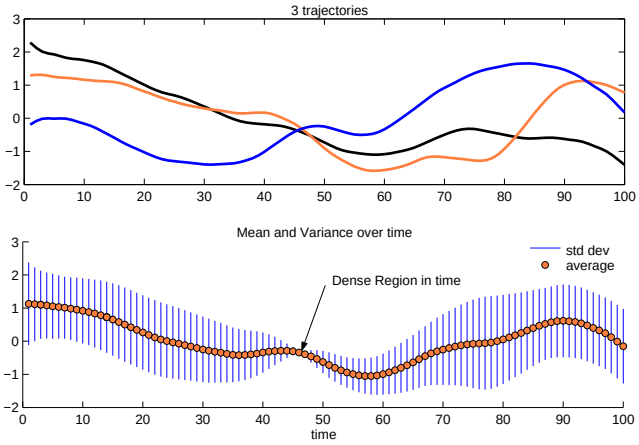

Figure 6: The mean and variance of three one-dimensional trajectories at every time point. Notice that in dense areas the variance is low.

The main reason for the slow performance of the greedy distance-based algorithm is that whenever two consecutive MBRs are merged, a total recomputation of the pairwise distances between all trajectories needs to be performed. One approach for speeding up this algorithm is based on the idea that at heavily clustered locations (i.e., time points where a large number of trajectories concentrate around the same area) one can afford to have coarser trajectory approximations since the total pairwise distance in these areas is very small in practice in the first place. This idea is illustrated in Figure 6.

To identify the heavily clustered time points efficiently one can compute for every time step an empirical mean and variance of the trajectories. Consecutive time instants with similar means and small variances intuitively contain clusters of trajectories. After such sequences of time instants have been found, the algorithm can start approximating the trajectories contained therein without substantially affecting the overall pairwise distance. In practice, the algorithm identifies the most clustered sequence of time instants, performs one merge operation, recomputes the mean and variance of the affected time points after the merge, and continues iteratively with the next candidate. Figure 7 shows a high-level description of the algorithm.

There are several issues that need to be discussed. First, notice that the value of a trajectory at a given time point is a two-dimensional vector. We define the mean and the variance of n elements $\boldsymbol{x}_i$ as

$$\boldsymbol{\mu} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i \qquad \text{Var} = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{\mu})^T \cdot (\boldsymbol{x}_i - \boldsymbol{\mu}),$$

where $\boldsymbol{x}^T$ is the transpose vector of $\boldsymbol{x}$. Notice that the

1. **Function** varianceSegmentation($T_1, \ldots, T_n$, K)
2. Start with one MBR per point per trajectory T_i
3. **foreach** time point t
4. Compute $\boldsymbol{\mu}(t), V(t)$
5. **while** (#MBRs $> K$)
6. $t^* \leftarrow$ denseAreaEstimate /* *We decide to merge two MBRs between time points* t^* *and* t^*+1 */*
7. $T^* \leftarrow$ leastVolumeIncrease
8. Merge T^*'s MBRs $[\cdot, t^*]$ and $[t^*+1, \cdot]$
9. Update the values $\mu(t)$ and $V(t)$ for the time instants spanned by the two merged MBRs
10. **end while**

Figure 7: Variance-based segmentation algorithm.

mean is a two-dimensional vector, while the variance is a real value.

The mean and variance can easily be computed if each trajectory is represented by a 1D point for a given time instant. Nevertheless, as trajectories are continually approximated with MBRs, for some time instants a number of trajectories will be represented by line segments (the projection of the MBR on that time instant; see Figure 8). While computing the mean $\sum_{i=1}^{n} T_i(t)/n$ when there are no rectangles is straightforward, computing the mean and the variance when some of the trajectories contain bounding rectangles at time t must involve both ends of the rectangle.

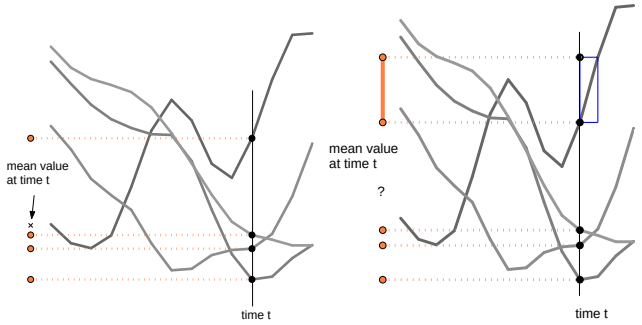

Figure 8: The definition of the mean and the variance have to be generalized to take into account the existence of the MBRs.

So, for the general case, we define the mean estimate to be the two-dimensional quantity

$$\boldsymbol{\mu}(t) = \frac{1}{2n} \sum_{i=1}^{n} \big(\boldsymbol{\ell}_i(t) + \boldsymbol{h}_i(t)\big),$$

and our variance estimate as the real value

$$V(t) = \frac{1}{2n} \sum_{i=1}^{n} \big((\boldsymbol{\ell}_i(t) - \boldsymbol{\mu}(t))^T \cdot (\boldsymbol{\ell}_i(t) - \boldsymbol{\mu}(t)) + (\boldsymbol{h}_i(t) - \boldsymbol{\mu}(t))^T \cdot (\boldsymbol{h}_i(t) - \boldsymbol{\mu}(t))\big).$$

After we have computed the means and variances of all the time points, we select a heavily clustered time point (line 7). Our estimator returns a time point t^* where the means $\boldsymbol{\mu}(t^*)$ and $\boldsymbol{\mu}(t^*+1)$ are close and the variances $V(t^*)$ and $V(t^*+1)$ are small. In detail, here is how we compute t^*. For each time point, define $\Delta\boldsymbol{\mu}(t) = \|\boldsymbol{\mu}(t) - \boldsymbol{\mu}(t+1)\|$. For some constant k (in our experiments we used $k \in \{1, \ldots, 50\}$) we compute the set $C^* = \{t_1^*, \ldots, t_k^*\}$ of candidates, which are the k time points with smallest corresponding $\Delta\boldsymbol{\mu}(t)$. Having computed the set C^*, we set t^* equal to

$$t^* = \arg\min_{t_i^*} \|(V(t_i^*), V(t_i^*+1))\|,$$

where we denote with $(V(t_i^*), V(t_i^*+1))$ the two-dimensional vector with elements $V(t_i^*)$ and $V(t_i^*+1)$ (see Figure 9).

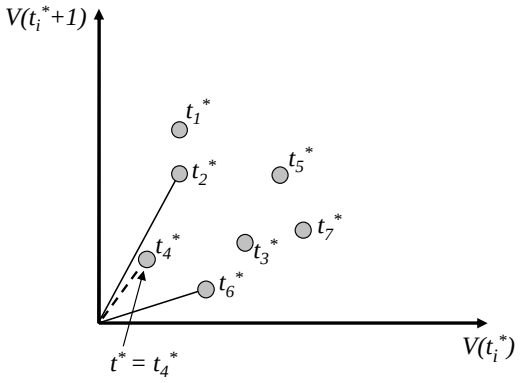

Figure 9: We select the time point t^* such that both $V(t^*)$ and $V(t^*+1)$ are small.

Having selected the time point t^*, we select at line 7 the trajectory whose MBRs we combine. Ideally we would like to merge the MBRs that will lead to the smallest decrease in the total pairwise distance. This approach, however, suffers from similar efficiency problems with the greedy approach: in the worst case we need to compute $O(mn)$ pairwise distances per MBR. A good compromise is to employ a local optimization criterion, namely we select to merge the two MBRs that will lead to a minimal total increase of volume.

As we show in the next theorem, the running time of the variance-based algorithm is much lower than the greedy algorithm.

THEOREM 3. *Assume that we execute the variance-based algorithm with n trajectories, m points per trajectory, and K MBRs, where $n \leq K \leq nm$. Then the time required to execute the algorithm is in the worst case $O(m^2 n \log n)$.*

PROOF. The initial computation of the mean and variances takes $O(mn)$ steps, while creating a heap with the pairwise differences of the means of consecutive time instants, which is useful in determining the denser time points, takes $O(m)$ steps. We perform the main loop of the algorithm $nm-K$ times. The dense area estimation can be computed in $k \log k$ steps. Choosing which trajectory to merge after the time point has been selected and updating the necessary data structure can be accomplished in $O(m \log n)$ steps (we keep a sorted list of all the trajectories for each time instance, but in every merge up to m lists might be updated resulting in time $O(m \log n)$), and updating the mean and variance of the affected time points in constant time for the means (we maintain the differences $\Delta\boldsymbol{\mu}(t)$ of the means, and only three of those values change), and $O(m)$ time for the variances. Finally, updating the list of mean differences requires $O(\log m)$ time. By accumulating all those steps, we conclude that in the worst case the time complexity of the algorithm is $O(m^2 n \log n)$. □

6. EMPIRICAL EVALUATION

We conduct a performance evaluation of the proposed algorithms using a number of real and synthetic datasets, with the objective of demonstrating the accuracy and performance of these techniques in practice. We compare all flavors of distance-based segmentation; the optimal dynamic programming algorithm, the greedy distance-based algorithm, and the variance-based segmentation techniques. We also include in our comparisons the local volume-based MBR segmentation [8, 15]. All algorithms were implemented in C++ and executed on a 3GHZ Pentium 4 with 1GB RAM.

6.1 Comparison to the optimal algorithm

The first experiment evaluates the qualitative affinity of all techniques to the optimal distance-based dynamic programming algorithm. We capture this by recording the sum of pairwise distance between the simplified trajectories (distance between the final MBRs).

For this experiment we utilize a large dataset pool, since the performance of each technique is highly dependent on the data characteristics. We use a number of real datasets from the UCR time-series data mining archive [9], which span a wide variety of areas, such as computer networks, medicine, environmental measurements, and more. Each dataset consists of 50 sequences with length of 512 points. We note that the datasets used in this experiment are 1D and not 2D. This was necessary in order to keep the running requirements of the optimal dynamic programming segmentation algorithm within the limits of the computational and storage capabilities of our computer testbed.

The performance of all strategies against the optimal approach is shown at the top of Figure 10. This figure depicts the results for each dataset separately. The y axis shows the sum of pairwise distances between all the objects, normalized by the total pairwise distance of the optimal algorithm. Numbers closer to 1 indicate better distance preservation. Each pair of sequences was simplified using 100 MBRs ($K = 100$). From the plots we can observe that in general the the greedy distance-based preserves closely the sum of pairwise distances achieved by the optimal, while the variance-based preserves a smaller amount of the pairwise distance sum. This is expected since the variance-based technique introduces many distance simplifications, in order to expedite the running performance. The distance affinity of the local volume-based algorithm stands between the two (non-optimal) distance-based algorithms.

Notice, that the previous experiment captures how close *absolute* distances are to the optimal algorithm. An even more meaningful experiment for data-mining operations, is how well *relative* distances are preserved. For example, if for the optimal algorithm the distances between objects A, B and C are *d(A,B) = 5* and *d(B,C) = 10* this will lead to a distance ratio of 1/2. In this setting, any algorithm X providing distance approximations of *d(A,B) = 1* and *d(B,C) = 2* might be weighted more favorably against another algorithm Y with respective distances of 3 and 4. Intuitively, the output of algorithm X can provide more similar behavior to the original data for clustering/classification operations, since relative relative object spacing will be affected the least (similar objectives are achieved by *data embeddings* [3]). The results for the relative distances are reported at the bottom of Figure 10 and in this experiment numbers approaching zero are closer to the optimal. One can observe that the variance based algorithm depicts better relative distance preservation than the volume-based for 25% of the datasets. These are primarily the datasets that contain

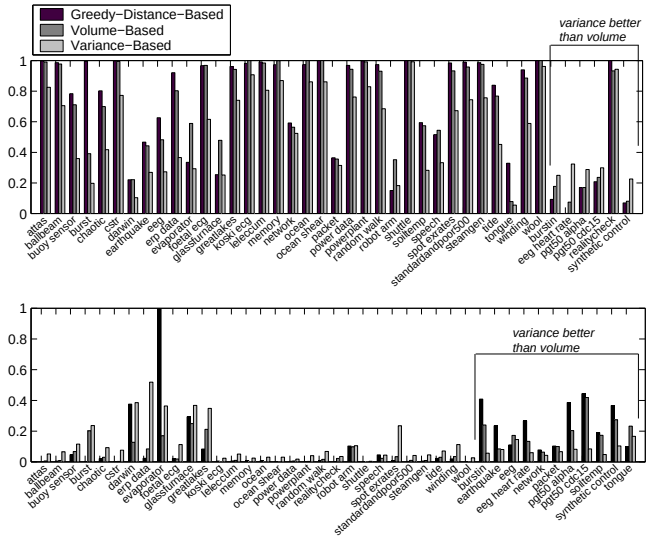

Figure 10: Affinity of heuristic algorithms to the optimal dynamic programming solution. *Top:* Closeness of absolute distances to optimal (numbers closer to 1 are better). *Bottom:* Closeness of relative distances to optimal (numbers closer to 0 are better).

multiple data bursts (e.g., earthquake, eeg, packet).

In general, the optimal and greedy distance-based algorithms present the best distance preservation. However, the optimal algorithm for any practical purpose is completely infeasible; our experiments for it took several days. The greedy-based runs in the order of minutes or hours and the variance- and volume-based require only seconds, for the data instances of this experiment. Each simplification typically detracts at least an order of magnitude in running time, which was also depicted with the complexity analysis of each algorithm. The most efficient algorithms are the local volume-based and the global variance-based. The first one exhibits better approximations for smooth datasets, while the second one is better for 'busier' datasets, where the global dataset view can provide a much better indication of which areas need to be simplified.

6.2 k-Nearest Neighbor Performance

Here we focus more closely on the performance of the the variance-based segmentation algorithm which is the only computationally feasible approach, and hence practical for real-world applications. We measure its performance on a k-Nearest-Neighbor (k-NN) retrieval experiment using a larger dataset instance, with synthetic datasets that simulate a road network of moving objects. We create multiple dataset instances with various object cardinalities (500, 2000, 4000 trajectories), where each object has length of 256 points.

The above dataset is utilized for evaluating the efficiency of the algorithm under k-NN search using the Euclidean distance metric. The segmented versions of the trajectories can speed up the search as follows. The trajectory segmentations represent a compressed version of the dataset, which essentially prioritize the retrieval of the raw trajectories from disk, by guiding the k-NN search.

Assume that the original trajectories are kept on disk but their simplifications (MBRs) are small enough to be stored in memory. The distance between the MBRs of two trajecto-

ries represent a *lower bound* of the original distance between the raw trajectories [15]. To identify the nearest neighbor one can compute a lower bound of the distances between the sequence approximations (given by each algorithm) and the query. The raw trajectories are then retrieved from disk in the order suggested by the lower bound (i.e., we examine the most likely candidates first). The best-so-far distance is potentially updated with every raw trajectory retrieved from disk. One can guarantee that the best match to the query is found when the currently examined compressed trajectory has a lower bound distance larger the distance of the best-so-far match. The number of retrieved matches from disk is an implementation invariant way of computing the efficiency of each segmentation method.

Note that the above search technique does not introduce any false dismissals, that is, the returned answer set will be exactly the same, as the k-NN search on the raw data. This is guaranteed by the lower bounding properties of the MBR approximation and is in accordance with the GEMINI indexing framework [2].

We perform the above k-NN search experiment using 100 different queries (not already found in the dataset) and in Figure 11 we plot the percentage of raw trajectories that are retrieved when searching for 20-NN, using different trajectory approximations (5, 10 and 20 MBRs per sequence). The above measure is an implementation independent way of measuring the efficiency of the representation. The experiment suggests that we consistently examine only a very small portion of the dataset, which reduces gracefully with finer trajectory approximations (i.e., use of more MBRs). While k-NN search on the raw data requires access to all trajectories on the disk, using the above simple technique we can prune from examination most of the trajectories.

The good k-NN search performance using distance-driven MBRs approximations is attributed to the fact that this type of segmentation essentially relaxes the distance in the already dense areas (small variance), and hence in practice it does not penalize the search performance, which is primarily impacted by the areas of large variance [7]. Additionally, we can observe that the preprocessing time required for segmenting the trajectories is kept in very realistic levels. The MBR generation time is shown in Figure 12 as a function of the dataset size.

We conducted the data-pruning experiment using also the volume-based segmentation technique (which shares the closest computational complexity with the variance based algorithm). The results are almost identical data pruning efficiency. Additional experiments are still needed for ascertaining whether these results can be generalized for other classes of datasets. Additionally it would be interesting to examine under what circumstances local techniques can be a viable alternative to global ones (see for example comparison of APCA vs SVD [10]).

As concluding remarks, with this empirical evaluation we have highlighted a lightweight version of global distance-based segmentation, which comes in the form of the variance-based segmentation. This method achieves significantly lower preprocessing times, while at the same time accomplishing distance relaxation only where needed, therefore not penalizing performance. This flavor of distance-based segmentation depicts excellent pruning power and therefore is suitable for any algorithm that utilizes k-NN search operations.

6.3 Application to DNA visualization

We conclude our experimental section with an interesting application and technique, which clearly highlights the importance of segmentation for manipulation and visualization of complex data. Specifically, we show how trajectory segmentation techniques can be used for visualizing and comparing the affinity of DNA strings.

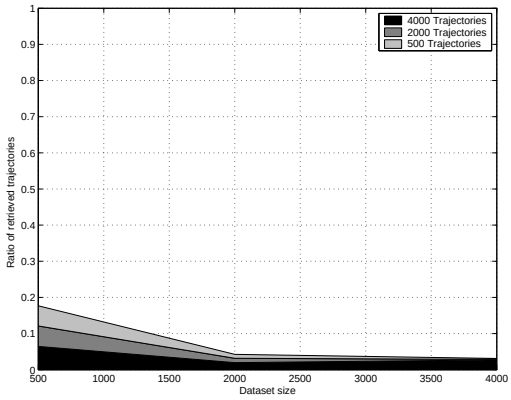

Figure 11: Evaluation of k-nearest neighbor queries. Percentage of retrieved disk resident trajectories for 20-NN search.

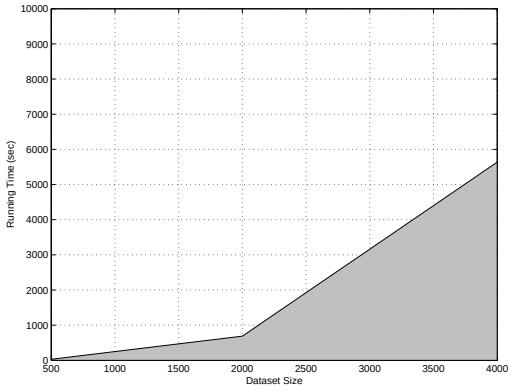

Figure 12: Segmentation time for K=20 MBRs with respect to dataset size.

Visual comparison of DNA symbol strings can be particularly troublesome to perform, because typical DNA datasets contains thousands of symbols. Humans cannot easily compare or visually represent bulk of text; our brains are much more efficient at comparing lines or shapes. Therefore, first we provide a technique for converting a DNA string into a two-dimensional trajectory. Given a string of length n drawn from the alphabet `A,T,C,G`, which we will denote as *speciesDNA*, we wish to convert it to a two-dimensional vector of length $n+1$, which we will denote as *speciesTrajectory*. We can use the following rule to build the trajectory vector:

$$speciesTrajectory(i) = speciesTrajectory(i-1) + \mathbf{B},$$

where $\mathbf{B}$ is a basis vector constructed as follows:

$$\mathbf{B} = \begin{cases} [0\ 1], \text{ if } speciesDNA(i) = \texttt{A} \\ [1\ 0], \text{ if } speciesDNA(i) = \texttt{T} \\ [0\ \text{-}1], \text{ if } speciesDNA(i) = \texttt{C} \\ [\text{-}1\ 0], \text{ if } speciesDNA(i) = \texttt{G}. \end{cases}$$

Example: Suppose that speciesTrajectory(1) = [0 0]. Thus, for the DNA string `AATCG`, we get the trajectory vector {[0 0],[0 1],[0 2],[1 2],[1 1],[0 1]}.

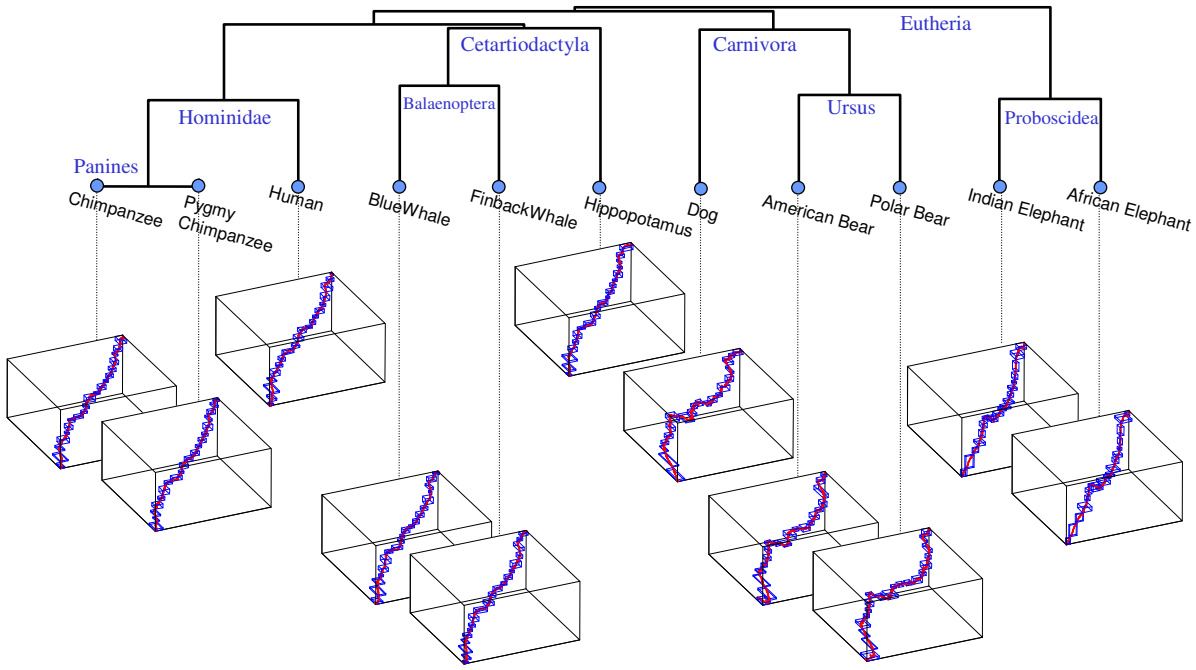

Figure 13: MtDNA dendrogram of 11 species. Transformation to trajectories and variance-based segmentation into 20 MBRs.

The resulting trajectories can be quite long, therefore it is more meaningful to be represented with fewer MBRs, which significantly aids their manipulation or on-screen depiction. Additionally, since the variance-based segmentation can accurately capture relative object distances, tasks such as classification, taxonomy categorization and dendrogram visualization, are expected to perform very effectively even on the simplified data. We demonstrate this later with an example. For simplicity we consider only mitochondrial DNA (mtDNA). mtDNA is passed on only from the mother during sexual reproduction, meaning that the mitochondria are clones. This means that there is little change in the mtDNA from generation to generation (i.e rare mutations), unlike nuclear DNA which changes by 50% each generation. This gives mtDNA a long *memory*.

Utilizing the mtDNA of 11 species (human, chimp, elephant, etc.), we create their respective trajectories using the technique described above and finally we segment each resulting trajectory into 20 MBRs using the variance-based algorithm. For this datasets the length of each mtDNA string consists of approximately 16000 symbols (with mtDNA of humans being 16,569 symbols long, and all other mammals mtDNA are within plus or minus 1% of this). The final MBR representation of every mtDNA, essentially represents a coarse signature of every symbol string. Notice that not only have we managed to represent the DNA string into a format that is amenable to visualization, but we have effectively *compressed* our dataset, since we have reduced 16000 points down to 20 MBRs. Every MBR can be represented by 4 numbers (lower and upper 2D points), therefore we have effectively compressed the original data by $(16000/(20 \times 4)) = 200$ times.

After computing the pairwise distance matrix between the MBR representation of all species, we create the (average linkage) dendrogram depicted on Figure 13. The first thing that one can notice is that even though the representation is highly compressed, the dendrogram correctly captures the taxonomy between the different species. A second observation on the figure is that, at first glance, the grouping of the hippo with whales might seem like a mistake. Intuitively the hippo should depict a greater affinity with the elephants. Interestingly, this is not the case; the hippos are more closely related to whales than to any other mammals! Whales and hippos diverged a mere 54 million years ago, whereas the whale/hippo group parted from the elephants about 105 million years ago. The group that includes hippo and whales/dolphins, but excludes all other mammals above is called Cetartiodactyla [14].

While this figure serves merely as a demonstration of the effectiveness of our MBR approach to capture structure in trajectories, we feel that such a representational transformation, combined with our MBR and other indexing tools and techniques could have great utility for mining large sequence collections. Recall that the full DNA sequence of a human is approximately 3 billion symbols long. Matching substrings, either within or between species is a computationally demanding task. While we are not suggesting this method to replace sophisticated string alignment methods, it could be used as a initial filtering step for finding promising candidate substrings.

7. CONCLUSIONS

In this work we motivated the need for global distance oriented segmentation techniques. We present different flavors of distance-based segmentation algorithms that operate at various scales of computational granularities. We introduce an optimal and a greedy version, and we show analytically and empirically that they are computationally impractical. However, we conclude the paper by presenting a variance-based hybrid variation that can provide an excellent compromise between running time and approximation quality.

8. REFERENCES

[1] B. Abarbanel and W. McNeely. FlyThru the Boeing 777. In *ACM Siggraph*, 1996.

[2] R. Agrawal, C. Faloutsos, and A. Swami. Efficient Similarity Search in Sequence Databases. *In Proc. of the 4th FODO*, pages 69–84, Oct. 1993.

[3] V. Athitsos, M. Hadjieleftheriou, G. Kollios, and S. Sclaroff. Query-sensitive embeddings. In *SIGMOD*, 2005.

[4] M. Cardle, M. Vlachos, S. Brooks, E. Keogh, and D. Gunopulos. Fast Motion Capture Matching with Replicated Motion Editing. In *ACM Siggraph*, 2003.

[5] P. Chan and M. Mahoney. Modeling Multiple Time Series for Anomaly Detection. In *ICDM*, 2005.

[6] J. D. Cohen, M. C. Lin, D. Manocha, and M. Ponamgi. I-COLLIDE: an interactive and exact collision detection system for large-scale environments. In *Symposium on Interactive 3D graphics*, 1995.

[7] A. P. deVries, N. Mamoulis, N. Nes, and M. L. Kersten. Efficient k-NN search on vertically decomposed data. In *SIGMOD*, 2002.

[8] M. Hadjieleftheriou, G. Kollios, V. Tsotras, and D. Gunopulos. Efficient indexing of spatiotemporal objects. In *Proc. of 8th EDBT*, 2002.

[9] E. Keogh. UCR time series data mining archive. http://www.cs.ucr.edu/∼eamonn/TSDMA/.

[10] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. of ACM SIGMOD*, pages 151–162, 2001.

[11] E. Keogh, T. Palpanas, V. Zordan, D. Gunopulos, and M. Cardle. Indexing Large Human-Motion Databases. In *VLDB*, 2004.

[12] R. Mann, A. D. Jepson, and T. El-Maraghi. Trajectory segmentation using dynamic programming. In *ICPR*, 2002.

[13] S. Rasetic, J. Sander, J. Elding, and M. A. Nascimento. A Trajectory Splitting Model for Efficient Spatio-Temporal Indexing. In *Proc. of VLDB*, 2005.

[14] B. M. Ursing and U. Arnason. Analyses of mitochondrial genomes strongly support a hippopotamus-whale clade. In *Proc. of the Royal Society of London, Series B, volume 265*, pages 2251–2255, 1998.

[15] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures. In *Proc. of SIGKDD*, 2003.

APPENDIX

Optimal Distance-Based Dynamic Programming Solution

Here we describe in detail the dynamic programming algorithm of Section 4 and we prove Theorem 1.

The algorithm maintains a matrix A, where $A[t, k, \ell_1, h_1, \ell_2, h_2]$ contains the value of the optimal way for assigning exactly k bounding rectangles to the two trajectories for time 0 up to and including time t (for $t \in \{0, 1, \ldots, m-1\}$, $k \in \{0, 1, \ldots, K\}$, and $\ell_1, h_1, \ell_2, h_2 \in \{1, 2, \ldots, R\}$), with the rightmost rectangle of trajectory i being on the range $[\ell_i, h_i]$. This value is $-\infty$ if there is no valid assignment of bounding rectangles (e.g., when $T_1(t) > h_1$).

We define the distance of two ranges to be:

$$d([l_1, r_1], [l_2, r_2]) = \min_{\substack{x_1 \in [l_1, r_1] \\ x_2 \in [l_2, r_2]}} d(x_1, x_2).$$

For the time step 0 we set the entries of A to:

- $A[0, k, \ell_1, h_1, \ell_2, h_2] = -\infty$, for $k \neq 2$;
- $A[0, 2, \ell_1, h_1, \ell_2, h_2] = -\infty$, for $T_1(0) \notin [\ell_1, h_1]$, or $T_2(0) \notin [\ell_2, h_2]$;
- $A[0, 2, \ell_1, h_1, \ell_2, h_2] = d([\ell_1, h_1], [\ell_2, h_2])$, for $T_1(0) \in [\ell_1, y_1]$, and $T_2(0) \in [\ell_2, y_2]$.

We now complete the table for increasing values of t. Assume that we know all the values $A[t-1, \cdot, \cdot, \cdot, \cdot, \cdot]$. We want to compute all the values $A[t, \cdot, \cdot, \cdot, \cdot, \cdot]$. For $T_1(t) \notin [\ell_1, h_1]$ we set $A[t, k, \ell_1, h_1, \ell_2, h_2] = -\infty$, for all k, ℓ_2, h_2. Similarly, for $T_2(t) \notin [\ell_2, h_2]$ we set $A[t, k, \ell_1, h_1, \ell_2, h_2] = -\infty$, for all k, ℓ_2, h_2.
Finally, for $T_1(t) \in [\ell_1, h_1]$ and $T_2(t) \in [\ell_2, h_2]$, we let for every $k, \ell_1, h_1, \ell_2 2, h_2$:

$$\begin{aligned} A[t, k, \ell_1, h_1, \ell_2, h_2] = d([\ell_1, h_1], [\ell_2, h_2]) + \\ \max\{A[t-1, k, \ell_1, h_1, \ell_2, h_2], \\ \max_{\hat{\ell_1}, \hat{h_1}} A[t-1, k-1, \hat{\ell_1}, \hat{h_1}, \ell_2, h_2], \\ \max_{\hat{\ell_2}, \hat{h_2}} A[t-1, k-1, \ell_1, h_1, \hat{\ell_2}, \hat{h_2}], \\ \max_{\hat{\ell_1}, \hat{h_1}, \hat{\ell_2}, \hat{h_2}} A[t-1, k-2, \hat{\ell_1}, \hat{h_1}, \hat{\ell_2}, \hat{h_2}]\}. \end{aligned}$$

The value of the optimal assignment of bounding rectangles is given by:

$$\max_{\hat{\ell_1}, \hat{h_1}, \hat{\ell_2}, \hat{h_2}} A[m-1, k, \hat{\ell_1}, \hat{h_1}, \hat{\ell_2}, \hat{h_2}].$$

In order to compute the optimal assignment we backtrack on array A, as is typical in dynamic programming.

First we show by induction that the algorithm computes an optimal assignment. Assume that for all $t' \leq t-1$, all the entries $A[t', \cdot, \cdot, \cdot, \cdot, \cdot]$ give the values of the optimal assignments of bounding rectangles. We show that our algorithm gives the value of the optimal assignment for the entries $A[t, \cdot, \cdot, \cdot, \cdot, \cdot]$.

Assume that this is not the case and that for some value of k, there is another optimal assignment that gives a higher value. Let V be the value of the assignment produced by our algorithm, and V^* be that of the optimal assignment. Assume that the coordinates of the rightmost rectangles (at time t) in the assignment given by our algorithm (the maximum of the assignments of $A[t, \cdot, \cdot, \cdot, \cdot, \cdot]$) are ℓ_1 and h_1 for the first trajectory, and ℓ_2 and h_2 for the second, and those corresponding to the optimal assignment are ℓ_1^*, h_1^*, ℓ_2^*, and h_2^*.

Let us focus on the optimal assignment. We look at the assignment of rectangles up to time $t-1$ in that one and assume that it uses k' rectangles (since we consider the case where the optimal uses k rectangles we have that $k-2 \leq k' \leq k$). By the induction hypothesis, an assignment at least as good as the optimal is the one corresponding to the entry

$$\begin{aligned} \max\{A[t-1, k, \ell_1^*, h_1^*, \ell_2^*, h_2^*], \\ \max_{\hat{\ell_1}, \hat{h_1}} A[t-1, k-1, \hat{\ell_1}, \hat{h_1}, \ell_2^*, h_2^*], \\ \max_{\hat{\ell_2}, \hat{h_2}} A[t-1, k-1, \ell_1^*, h_1^*, \hat{\ell_2}, \hat{h_2}], \\ \max_{\hat{\ell_1}, \hat{h_1}, \hat{\ell_2}, \hat{h_2}} A[t-1, k-2, \hat{\ell_1}, \hat{h_1}, \hat{\ell_2}, \hat{h_2}]\}. \end{aligned}$$

Since our algorithm examines all those cases it would have found it, so we end with a contradiction. Therefore, eventually the algorithm finds the optimal assignment.

For the space complexity, notice that the size of A is $m \times K \times R \times R \times R \times R$, while for the time complexity, notice that in order to compute each entry of A we need to perform at most $O(R^4)$ operations.