Published in final edited form as:

ACM Trans Knowl Discov Data. 2013 September ; 7(3): .

Addressing Big Data Time Series: Mining Trillions of Time Series Subsequences Under Dynamic Time Warping

THANAWIN RAKTHANMANON
University of California Riverside and Kasetsart University

BILSON CAMPANA, **ABDULLAH MUEEN**
University of California Riverside

GUSTAVO BATISTA,
University of São Paulo

BRANDON WESTOVER
Brigham and Women's Hospital

QIANG ZHU, **JESIN ZAKARIA**, **EAMONN KEOGH**
University of California Riverside

Abstract

Most time series data mining algorithms use similarity search as a core subroutine, and thus the time taken for similarity search is the bottleneck for virtually all time series data mining algorithms, including classification, clustering, motif discovery, anomaly detection, and so on. The difficulty of scaling a search to large datasets explains to a great extent why most academic work on time series data mining has plateaued at considering a few millions of time series objects, while much of industry and science sits on billions of time series objects waiting to be explored. In this work we show that by using a combination of four novel ideas we can search and mine massive time series for the first time. We demonstrate the following unintuitive fact: in large datasets we can exactly search under Dynamic Time Warping (DTW) much more quickly than the current state-of-the-art *Euclidean distance* search algorithms. We demonstrate our work on the largest set of time series experiments ever attempted. In particular, the largest dataset we consider is larger than the combined size of all of the time series datasets considered in all data mining papers ever published. We explain how our ideas allow us to solve higher-level time series data mining problems such as motif discovery and clustering at scales that would otherwise be untenable. Moreover, we show how our ideas allow us to efficiently support the uniform scaling distance

thanawin.r@ku.ac.th;.
Rakthanmanon has a dual affiliation: University of California Riverside and Kasetsart University.
Authors' addresses: T. Rakthanmanon, Department of Computer Engineering, Kasetsart University, Thailand; B. Campana, A. Mueen, Q. Zhu, J. Zakaria, and E. Keogh, Department of Computer Science and Engineering, University of California Riverside; G. Batista, Instituto de Ciências Matemáticas e de Computaçã, University of São Paulo; B. Westover, Brigham and Women's Hospital.

measure, a measure whose utility seems to be underappreciated, but which we demonstrate here. In addition to mining massive datasets with up to one trillion datapoints, we will show that our ideas also have implications for real-time monitoring of data streams, allowing us to handle much faster arrival rates and/or use cheaper and lower powered devices than are currently possible.

General Terms:

Algorithms; Experimentation

Additional Key Words and Phrases:

Time series; similarity search; lower bounds

1. INTRODUCTION

Time series data is pervasive across almost all human endeavors, including medicine, finance, science, and entertainment. As such, it is hardly surprising that time series data mining has attracted significant attention and research effort. Most time series data mining algorithms require similarity comparisons as a subroutine, and in spite of the consideration of dozens of alternatives, there is increasing evidence that the classic Dynamic Time Warping (DTW) measure is the best measure in most domains [Ding et al. 2008].

It is difficult to overstate the ubiquity of DTW. It has been used in robotics, medicine [Chadwick et al. 2011], biometrics, music/speech processing [Adams et al. 2005; Muller 2009; Zhang and Glass 2011], climatology, aviation, gesture recognition [Alon et al. 2009; Wobbrock et al. 2007], user interfaces [Hsiao et al. 2005; Laerhoven et al. 2009; Pressly 2008; Wobbrock et al. 2007], industrial processing, cryptanalysis [Dupasquier and Burschka 2011], mining of historical manuscripts [Huber-Märk et al. 2011], geology, astronomy [Keogh et al. 2009; Rebbapragada et al. 2009], space exploration, wildlife monitoring, and so on.

As ubiquitous as DTW is, we believe that there are thousands of research efforts that would like to use DTW, but find it too computationally expensive. For example, consider the following: “Ideally, dynamic time warping would be used to achieve this, but due to time constraints… ” [Chadwick et al. 2011]. Likewise, [Alon et al. 2009] bemoans DTW is “still too slow for gesture recognition systems,” and [Adams et al. 2005] notes, even “a 30 fold speed increase may not be sufficient for scaling DTW methods to truly massive databases.” As we shall show, our subsequence search suite of four novel ideas (called the *UCR suite*) removes all of these objections. We can reproduce all of the experiments in all of these papers in well under a second. We make an additional claim for our UCR suite that is almost certainly true, but very hard to prove, given the variability in how search results are presented in the literature. We believe our exact DTW sequential search is much faster than any current approximate search or exact indexed search. In a handful of papers the authors are explicit enough with their experiments to see this is true. Consider Papapetrou et al. [2011], in which the authors introduce a technique that can answer queries of length 1000 under DTW with 95% accuracy, in a random walk dataset of one million objects in 5.65

seconds. We can exactly search this dataset in 3.8 seconds (on a very similar machine). Likewise, a recent paper that introduced a novel inner product-based DTW lower bound greatly speeds up exact subsequence search for a wordspotting task in speech. The authors state: "the new DTW-KNN method takes approximately 2 minutes" [Zhang and Glass 2011]; however, we can reproduce their results in less than a second. An influential paper on gesture recognition on multitouch screens laments that "DTW took 128.26 minutes to run the 14,400 tests for a given subject's 160 gestures" [Wobbrock et al. 2007]. However, we can reproduce these results in less than three seconds.

Our goal in this work is not just to demonstrate that we can search massive time series datasets more quickly than the current state-of-the-art approach. We also hope to enable the community to do higher level analysis of massive datasets by freely providing all of our code and data. Moreover, at least some of the four novel ideas we introduce here may have implications for problems beyond similarity search and beyond time series. For example, other distance measures may benefit from optimizing the order of evaluation in conjunction with admissible early termination, an idea we introduce as *reordering early abandoning* (cf. Section 4.2.2). Likewise the idea of cascading lower bounds (cf. Section 4.2.4) may be useful for other distance measures that are computationally expensive and have many proposed lower bounds of different time complexities/tightnesses. Both the Earth Mover's Distance and the String Edit Distance are tentative examples.

1.1. Millions, Billions, and Trillions, A Discussion of a Scale

Since we search a trillion objects in this work and to our knowledge, such a large dataset has never been considered in a data mining/database paper before, we will take the time to explicitly discuss this number. By a trillion, we mean the short scale version of the word [Guitel 1975], one million million, or 10^{12}, or 1,000,000,000,000.

If we have a single time series T of length one trillion, and we assume it takes eight bytes to store each value, it will require 7.2 terabytes to store. If we sample an electrocardiogram at 256Hz, a trillion datapoints would allow us to record 123 years of data, or every single heartbeat of the longest lived human [Whitney 1997].

As large as a trillion is, there are thousands of research labs and commercial enterprises that have this much data. For example, many research hospitals have trillions of datapoints of EEG data, NASA Ames has tens of trillions of datapoints of telemetry of domestic flights, the Tennessee Valley Authority (a power company) records a trillion datapoints every four months, and so on.

1.2. Explicit Statement of Our Assumptions

Our work is predicated on several assumptions that we will now enumerate and justify.

1.2.1. Time Series Subsequences Must Be Normalized.—In order to make meaningful comparisons between two time series, both must be normalized. We define normalization more formally later, but for now it is sufficient to consider it as the process of making two time series commensurate by transforming them to some canonical scale. While there are several such transformations possible, the vast majority of the literature uses Z-

normalization. Z-normalization shifts and scales the time series such that the mean is zero and the standard deviation is one. In Matlab, this can be achieved by the single line:

$$T = (T - \text{mean}(T))/\text{std}(T)$$

It is critical to avoid a common misunderstanding. We must normalize each subsequence before making a comparison; it is not sufficient to normalize the entire dataset, because then when individual subsequences are extracted, they will not be normalized.

While the need for normalization may seem intuitive, and was empirically demonstrated a decade ago in a widely cited paper [Keogh and Kasetty 2003], many research efforts do not seem to realize this. This is critical because some speedup techniques only work on the unnormalized data; thus the contributions of these research efforts may be largely nullified [Chen et al. 2009; Papapetrou et al. 2011]. To demonstrate the necessity of normalization we will begin with an intuitive and visual example, before considering some objective experiments.

In Figure 1(left) we show two examples of snippets of electrocardiograms (ECG) of individuals. Note that the mean of the trace labeled chf11 drifts up and down, a commonly observed phenomenon known as *wandering baseline*. This wandering baseline may be caused by patient movement, dirty lead wires/electrodes, loose electrodes, and so on. Note, however, that it does not have any medical significance; it is considered to be an artifact (in electrocardiography, an *artifact* is used to indicate something that is not “heart-made”). In Figure 1(right) we show what the individual heartbeats look like when we extract them without any attempt to normalize them.

Note that the two individuals’ beats in Figure 1(right) would be trivially easy to classify or cluster by eye. How would Euclidean distance do? To test this, in Figure 2 we performed a single-linkage hierarchical clustering of some randomly chosen exemplars from both individuals. We consider both the normalized and the nonnormalized versions of the data.

The results are very clear. The similarity measurements of unnormalized data, when used for clustering, give the wrong results, on a problem that could not be simpler. In contrast, working with the normalized data gives the correct results. Note that this experiment is not contrived in any way. Using DTW does not help, a different random subset of heartbeats does not help, and choosing other individuals’ ECG traces will not help, as virtually all ECG traces have wandering baselines. In fact, the two examples shown were chosen because they have relatively little wander, and are thus easer to plot. In the domain of ECG, we must normalize the data.

To show more quantitatively the effect of not normalizing data, let us consider the classic Gun/NoGun classification problem which has been in the public domain for nearly a decade. The data, which as shown in Figure 3(center) is extracted from a video sequence, was Z-normalized. The problem has a 50/150 train/test split and a DTW one-nearest-neighbor classifier achieves an error rate of 0.087.

Suppose the data had not been normalized. As shown in Figure 3(left) and Figure 3(right), we can simulate this by adding a tiny amount of scaling/offset to the original video. In the first case we randomly change the offset of each time series by ±10%, and in the second case we randomly change the scale (amplitude) by ±10%. The new one-nearest-neighbor classifier error rates, averaged over 1000 runs, are 0.326 and 0.193, respectively, significantly worse than the normalized case.

It is important to recognize that these tiny changes we made are completely dwarfed by changes we might expect to see in a real-world deployment. The apparent scale can be changed by the camera zooming, by the actor standing a little closer to the camera, or by an actor of a different height. The apparent offset can be changed by this much by the camera tilt angle, or even by the actor wearing different shoes.

While we did this experiment on a visually intuitive example, all forty-five datasets in the UCR archive increase their error rate by at least 50% if we vary the offset and scale by just ± 5%.

1.2.2. Dynamic Time Warping Is the Best Measure.—It has been suggested many times in the literature that the problem of time series data mining scalability is only due to DTW's oft-touted lethargy, and that we could solve this problem by using some other distance measure. As we shall later show, this is not the case. In fact, as we shall demonstrate, our optimized DTW search is much faster than all current Euclidean distance searches. Nevertheless, the question remains, is DTW the right measure to speed up? Dozens of alternative measures have been suggested. However, recent empirical evidence strongly suggests that none of these alternatives routinely beats DTW. When put to the test on a collection of forty datasets, the very best of these measures are sometimes a little better than DTW and sometimes a little worse [Ding et al. 2008]. In general, the results are consistent with these measures being minor variants, or flavors, of DTW (although they are not typically presented this way). In summary, after an exhaustive literature search of more than 800 papers [Ding et al. 2008], we are not aware of any distance measure that has been shown to outperform DTW by a statistically significant amount on reproducible experiments [Ding et al. 2008; Keogh and Kasetty 2003]. Thus, DTW is the measure to optimize (recall that DTW subsumes Euclidean distance as a special case).

1.2.3. Arbitrary Query Lengths Cannot Be Indexed.—If we know the length of queries ahead of time we can mitigate at least some of the intractability of search by indexing the data [Assent et al. 2008; Fu et al. 2008; Shieh and Keogh 2008]. Although to our knowledge noone has built an index for a trillion real-valued objects (Google only indexed a trillion Web pages as recently as 2008), perhaps this could be done.

However, what if we do not know the length of the queries in advance? At least two groups have suggested techniques to index arbitrary length queries [Kahveci and Singh 2004; Lim et al. 2007]. Both methods essentially build multiple indexes of various lengths, and at query time search the shorter and longer indexes, interpolating the results to produce the nearest neighbor produced by a virtual index of the correct length. This is an interesting idea, but it is hard to imagine it is the answer to our problem. Suppose we want to support queries in the

range of, say, 16 to 4096. We must build indexes that are not too different in size, say, MULTINDEX-LENGTHS ={16, 32,64, …, 1024, 2048, 4096}.[1] However, for time series data, the index is typically about one-tenth the size of the data [Ding et al. 2008; Kahveci and Singh 2004]. Thus, we have doubled the amount of disk space we need. Moreover, if we are interested in tackling a trillion data objects, we clearly cannot fit any index in the main memory, much less all of them, or any two of them.

There is an underappreciated reason why this problem is so hard; it is an implication of the need for normalization. Suppose we have a query Q of length 65, and an index that supports queries of length 64. We search the index for $Q_{[1:64]}$ and find that the best match for it has a distance of, say, 5.17. What can we say about the best match for the full Q? The answer is surprisingly little: 5.17 is neither an upper bound nor a lower bound to the best match for Q. This is because we must renormalize the subsequence when moving from $Q_{[1:64]}$ to the full Q. If we do not normalize any data, the results are meaningless (Section 1.2.1), and the idea might be faster than sequential search. However, if we normalize the data we get so little information from indexes of the wrong length that we are no better off than sequential search.

In summary, there are no known techniques to support similarity search of arbitrary lengths once we have datasets in the billions.

1.2.4. There Exist Data Mining Problems That We Are Willing to Wait Some Hours to Answer.—

This point is almost self-evident. If a team of entomologists has spent three years gathering 0.2 trillion datapoints [Shieh and Keogh 2008], or astronomers have spent billions of dollars to launch a satellite to collect one trillion datapoints of star-light curve data per day [Keogh et al. 2009], or a hospital charges $34,000 for a daylong EEG session to collect 0.3 trillion datapoints (Section 5.2) [Mueen et al. 2011], then it is not unreasonable to expect that these groups would be willing to spend hours of CPU time to glean knowledge from their data.

2. RELATED WORK

Our review of related work on time series indexing is necessarily superficial, given the vast amount of work on the topic and our page limits. Instead, we refer the interested reader to two recent papers [Ding et al. 2008; Papapetrou et al. 2011], which have comprehensive reviews of existing work. It has now become common (although not yet routine) to see papers indexing/mining datasets with millions of objects. For example, Jegou et al. [2010] have demonstrated very fast approximate main memory search of 10 million images. However, this work and much of the current work that addresses multimillion object datasets focus on approximate search, whereas we are only considering exact search here. Moreover, we are interested in datasets that are five to six orders of magnitude larger than anything else considered in the literature [Ding et al. 2008]. Thus, comparisons to related work are very difficult to do meaningfully.

[1]This collection of sizes is very optimistic. The step size should be at most 100, creating two orders of magnitude space overhead.

Most techniques for speeding up similarity search exploit the triangular inequality. The idea is to precompute the distances between objects in the database and then later at query time exploit the fact that if the distance between the query Q and an object O_i in the database is calculated, then any object O_j can be pruned from further consideration if $|d(Q,O_i) - d(O_i,O_j)|$ is not less than *best-so-far*. Precomputing and storing all of the distances between objects in the database is impractical for large datasets; however, most spatial access methods can approximate doing this by grouping the similar objects (typically in Minimum Bounding Rectangles (MBRS)) and precomputing only the distances between MBRs. Unfortunately, these ideas do not help for DTW, which is not a metric and therefore does not obey the triangular inequality. Moreover, it does not help for Euclidean distance unless we know the length of the query ahead of time, an assumption we are explicitly avoiding. Thus, the vast majority of work on speeding up similarity search for time series (see Ding et al. [2008] and the references therein) do not help us.

Finally, we note that DTW is at least superficially similar to string edit distance [Chen and Ng 2004; Masek and Paterson 1980]. The reader may wonder if any speedup techniques for the latter can help us here. We believe the answer is no. Moving from the discrete to the real-valued creates unique problems. For example, as we will show, similarity search under DTW spends more time Z-normalizing the data than computing the actual DTW; however, there is no analogue of Z-normalizing for strings. Moreover, when working with strings one can use suffix trees, hashing, equality tests, and a host of other techniques that simply are not defined for real-valued data.

3. BACKGROUND AND NOTATION

3.1. Definitions and Notations

We begin by defining the data type of interest: time series.

Definition 1. A *Time Series* T is an ordered list of real-valued numbers: $T = t_1, t_2, \ldots, t_m$.

While the source data is one long time series with m datapoints, we ultimately wish to compare it to shorter regions called subsequences.

Definition 2. A *subsequence* $T_{i,k}$ of a time series T is a shorter time series of length k, which starts from position i. Formally, $T_{i,k} = t_i, t_{i+1}, .., t_{i+k-1}$, $1 \leq i \leq m - k + 1$.

Where there is no ambiguity, we may refer to subsequence $T_{i,k}$ as C, as in a candidate match to a query Q. We denote $|Q|$ as n. Moreover, a subsequence normalized to have a mean of zero and a standard derivation of one is called a *Z-normalized subsequence*.

Definition 3. The *Euclidean distance (ED)* between two Z-normalized subsequences Q and C, where $|Q|=|C|$, is defined as:

$$ED(Q, C) \sqrt{\sum_{i=1}^{n} (q_i - c_i)^2}.$$

We illustrate these definitions in Figure 4.

The Euclidean distance, which is a one-to-one mapping of the two sequences, can be seen as a special case of DTW, which allows a one-to-many alignment, as illustrated in Figure 5.

To align two sequences using DTW, an n-by-n matrix is constructed, with the (i^{th}, j^{th}) element of the matrix being the Euclidean distance $d(q_i, c_j)$ between the points q_i and c_j.

A warping path P is a contiguous set of matrix elements that defines a mapping between Q and C. The t^{th} element of P is defined as $p_t = (i, j)_t$, so we have:

$$P = p_1, p_2, \ldots, p_t, \ldots, p_T \quad n \leq T \leq 2n - 1.$$

The warping path that defines the alignment between the two time series is subject to several constraints. For example, the warping path must start and finish in diagonally opposite corner cells of the matrix, the steps in the warping path are restricted to adjacent cells, and the points in the warping path must be monotonically spaced in time. In addition, virtually all practitioners using DTW also constrain the warping path in a global sense by limiting how far it may stray from the diagonal [Ding et al. 2008; Papapetrou et al. 2011]. A typical constraint is the Sakoe-Chiba Band, which states that the warping path cannot deviate more than R cells from the diagonal [Ding et al. 2008; Papapetrou et al. 2011; Sakurai et al. 2007].

We are finally in a position to define the problem we wish to solve. For searching one nearest neighbor under DTW, given a long time series T, and a user-supplied query Q, where $m = |T| >> |Q| = n$. We wish to find the subsequence $T_{i:i+n-1}$, such that DTW(Q, $T_{i:i+n-1}$) is minimized. In other words:

$$\exists_i \forall_k \mathrm{DTW}\left(Q, T_{i:i+n-1}\right) \leq \mathrm{DTW}\left(Q, T_{k:k+n-1}\right), \; 1 \leq i, k \leq m - n + 1.$$

The one nearest neighbor ED search requires just the substitution of ED for DTW. The generalizations of one nearest neighbor to both K nearest neighbor and range search are trivial, and are omitted for clarity.

4. ALGORITHMS

4.1. Known Optimizations

We begin by discussing previously known optimizations of sequential search under ED and/or DTW.

4.1.1. Using the Squared Distance.—Both DTW and ED have a square root calculation. However, if we omit this step, it does not change the relative rankings of nearest neighbors, since both functions are monotonic and concave. Moreover, the absence of the square root function will make later optimizations possible and easier to explain. Note that this is only an internal change in the code; the user can still issue range queries with the original units, as the code simply internally squares the desired value, does the search, and after finding the qualifying objects, takes the square root of the distances for the qualifying objects and presents the answers to the user.

Where there is no ambiguity, we will still use “DTW” and “ED”; however, the reader may assume we mean the squared versions of them.

4.1.2. Lower Bounding.—A classic trick to speed up sequential search with an expensive distance measure such as DTW is to use a cheap-to-compute lower bound to prune off unpromising candidates [Ding et al. 2008; Keogh et al. 2009]. Figure 6 shows two such lower bounds, one of which we have modified.

The original definition of LB_Kim also uses the distances between the maximum values from both time series and the minimum values between both time series in the lower bound, making it $O(n)$. However, for normalized time series, these two extra values tend to be tiny so it does not pay to compute them, and ignoring them allows the bound to be $O(1)$, a fact we will exploit in the following. The LB_Keogh bound is well-documented elsewhere; for brevity we ask the unfamiliar reader to refer to Ding et al. [2008], Fu et al. [2008], and Keogh et al. [2009] for a review. However, for completeness we present a brief review. The upper envelope U and the lower envelope L of subsequence Q are defined in Keogh and Ratanamahatana [2005] as:

$$U_i = max(Q_{i-r}, Q_{i-r+1}, \ldots, Q_{i+r})$$
$$L_i = min(Q_{i-r}, Q_{i-r+1}, \ldots, Q_{i+r}),$$

where r is the size of the warping window, and the LB_Keogh lower bound is simply the distance from the closer of the two envelopes to another subsequence, C.

$$LB_{keogh} = \sqrt{\sum_{i=1}^{n} \begin{cases} (C_i - U_i)^2 & \text{if } C_i > U_i \\ (C_i - L_i)^2 & \text{if } C_i > L_i \\ 0 & \text{otherwise} \end{cases}}.$$

Since its introduction a decade ago, the LB_Keogh lower bound has been the cornerstone of most efforts to scale DTW similarity search [Ding et al. 2008; Fu et al. 2008;Keogh et al. 2009].

4.1.3. Early Abandoning of ED and LB_Keogh.—During the computation of the Euclidean distance or the LB_Keogh lower bound, if we note that the current sum of the squared differences between each pair of corresponding datapoints exceeds the *best-so-far*, then we can stop the calculation, secure in the knowledge that the exact distance or lower bound, had we calculated it, would have exceeded the *best-so-far*, as in Figure 7.

This simple idea that one can abandon an unpromising calculation the moment one can be sure it could not produce a result better than the best-so-far result has a long tradition in machine learning, artificial intelligence and image processing [Bei and Gray 1985; Cheng et al. 1984; McNames 2000].

4.1.4. Early Abandoning of DTW.—If we have computed a full LB_Keogh lower bound, but we find that we must calculate the full DTW, there is still one trick left up our sleeves. We can incrementally compute the DTW from left to right, and as we incrementally calculate from 1 to K, we can sum the partial DTW accumulation with the LB_Keogh contribution from K + 1 to n. Figure 8 illustrates this idea.

This sum of $min_j(\text{DTW}(Q_{1:K}, C_{1:K+j})) + LB\_Keogh(Q_{K+r+1:n}, C_{K+r+1:n})$ is a lower bound to the true DTW distance ($\text{DTW}(Q_{1:n}, C_{1:n})$), where r is the size of the warping windows. Moreover, with careful implementation, the overhead costs are negligible. If at any time this lower bound exceeds the *best-so-far* distance we can admissibly stop the calculation and prune this C.

4.1.5. Exploiting Multicores.—It is important to note that while we can get essentially linear speedup using multicores, the software improvements we will present in the next section completely dwarf the improvements gained by multicores. As a concrete example, a recent paper shows that a search of a time series of length 421,322 under DTW takes "3 hours and 2 minutes on a single core. The (8-core version) was able to complete the computation in 23 minutes" [Srikanthan et al. 2011]. However, using our ideas, we can search a dataset of this size in just under one second on a single core. Nevertheless, as it is simple to port to the now ubiquitous multicores, we consider them in the following.

4.2. Novel Optimizations: The UCR Suite

We are finally in a position to introduce our four original optimizations of search under ED and/or DTW.

4.2.1. Early Abandoning Z-Normalization.—To the best of our knowledge, noone has ever considered optimizing the normalization step. This is surprising, since it takes only slightly longer than computing the Euclidean distance itself.

Our insight here is that we can interleave the early abandoning calculations of Euclidean distance (or LB_Keogh) with the online Z-normalization. In other words, as we are incrementally computing the Z-normalization, we can also incrementally compute the Euclidean distance (or LB_Keogh) of the same datapoint. Thus, if we can early abandon, we are pruning not just distance calculation steps as in Section 4.1.3, but also normalization steps.

The fact that the mean and standard deviation of a stream of numbers can be incrementally calculated and maintained has long been exploited in computer science [Chan et al. 1983; Ling 1974]. Such algorithms are sometimes called "one pass" algorithms. However, to our knowledge, this is the first work to show that we can interleave early abandoning calculations with the incremental mean and standard deviation calculations.

Recall that the mean and standard deviation of a sample can be computed from the sums of the values and their squares. Therefore, it takes only one scan through the sample to compute the mean and standard deviation, using the following equations.

$$\mu = \frac{1}{m}\sum x_i \quad \sigma^2 = \frac{1}{m}\sum x_i^2 - \mu^2.$$

In similarity search, every subsequence needs to be normalized before it is compared to the query (Section 1.2.1). The mean of the subsequence can be obtained by keeping two running sums of the long time series, which have a lag of exactly m values. The sum of squares of the subsequence can be similarly computed. The formulas are given here for clarity.

$$\mu = \frac{1}{m}\left(\sum_{i=1}^{k} x_i - \sum_{i=1}^{k-m} x_i\right) \quad \sigma^2 = \frac{1}{m}\left(\sum_{i=1}^{k} x_i^2 - \sum_{i=1}^{k-m} x_i^2\right) - \mu^2$$

The high-level outline of the algorithm is presented in Table I.

Note the online normalization in line 11 of the algorithm, which allows the early abandoning of the distance computation in addition to the normalization. In the algorithm, we use a circular buffer (X) to store the current subsequence being compared with the query Q.

One potential problem of this method of maintaining the statistics is the accumulation of the floating-point error [Goldberg 1991]. The effect of such error accumulation is more profound if all of the numbers are positive, as in our case with the sum of squares. With the "mere" millions of datapoints, which the rest of the community has dealt with, this effect is negligible; however, when dealing with billions of datapoints it will affect the answer. Our simple solution is that once every one million subsequences, we force a complete Z-normalization to flush out any accumulated error.

4.2.2. Reordering Early Abandoning.—In the previous section, we saw that the idea of early abandoning discussed in Section 4.1.3 can be generalized to the Z-normalization step. In both cases, we assumed that we incrementally compute the distance/normalization from left to right. Is there a better ordering?

Consider Figure 9(left), which shows the normal left-to-right ordering in which the early abandoning calculation proceeds. In this case nine of the thirty-two calculations were performed before the accumulated distance exceeded b and we could abandon. In contrast, Figure 9(right) uses a different ordering and was able to abandon earlier, with just five of the thirty-two calculations.

This example shows what is obvious: on a query-by-query basis, different orderings produce different speedups. However, we want to know if there is a universal optimal ordering that we can compute in advance. This seems like a difficult question because there are $n!$ possible orderings to consider.

We conjecture that the universal optimal ordering is to sort the indices based on the absolute values of the Z-normalized Q. The intuition behind this idea is that the value at Q_i will be compared to many C_i's during a search. However, for subsequence search, with Z-normalized candidates, the distribution of many C_i's will be Gaussian, with a mean of zero.

Thus, the sections of the query that are farthest from the mean will on average have the largest contributions to the distance measure.

To see if our conjecture is true we took the heartbeat discussed in Section 5.4 and computed its full Euclidean distance to a million other randomly chosen ECG sequences. With the conceit of hindsight we computed what the best ordering would have been. For this we simply take each C_i and sort them, largest first, by the sum of their contributions to the Euclidean distance. We compared this empirically optimal ordering with our predicted ordering (sorting the indices on the absolute values of Q) and found the rank correlation is 0.999. Note that we can use this trick for both ED and LB_Keogh, and we can use it in conjunction with the early abandoning Z-normalization technique (Section 4.2.1). The proof of the optimal ordering for ED is provided in Section 5.6.

4.2.3. Reversing the Query/Data Role in LB_Keogh.—Normally the LB_Keogh lower bound discussed in Section 4.1.2 builds the envelope around the query, a situation we denote $LB\_KeoghEQ$ for concreteness, and illustrate in Figure 10(left). This only needs to be done once, and thus saves the time and space overhead that we would need if we built the envelope around each candidate instead, a situation we denote $LB\_KeoghEC$.

However, as we show in the next section, we can selectively calculate $LB\_KeoghEC$ in a just-in-time fashion, only if all other lower bounds fail to prune. This removes space overhead, and as we will see, the time overhead pays for itself by pruning more full DTW calculations. Note that in general, $LB\_KeoghEQ \neq LB\_KeoghEC$, and that on average each one is larger about half the time.

4.2.4. Cascading Lower Bounds.—One of the most useful ways to speed up time series similarity search is to use lower bounds to admissibly prune off unpromising candidates [Ding et al. 2008; Fu et al. 2008]. This has led to a flurry of research on lower bounds, with at least eighteen proposed for DTW [Adams et al. 2005; Ding et al. 2008; Keogh et al. 2009; Kim et al. 2001; Sakurai et al. 2005; Yi et al. 1998; Zhang and Glass 2011; Zinke and Mayer 2006]. In general, it is difficult to state definitively which is the best bound to use, since there is a tradeoff between the tightness of the lower bound and how fast it is to compute. Moreover, different datasets and even different queries can produce slightly different results. However, as a starting point, we implemented all published lower bounds and tested them on fifty different datasets from the UCR archive, plotting the (slightly idealized for visual clarity) results in Figure 11. Following the literature [Keogh et al. 2009], we measured the *tightness* of each lower bound as $LB(A, B)/DTW(A, B)$ over 100,000 randomly sampled subsequences A and B of length 256.

The reader will appreciate that a necessary condition for a lower bound to be useful is for it to appear on the “skyline” shown with a dashed line; otherwise there exists a faster-to-compute bound that is at least as tight, and we should use that instead. Note that the early abandoning DTW discussed in Section 4.1.4 is a special case in that it produces a spectrum of bounds, as at every stage of computation it is incrementally computing the DTW until the last computation gives the final true DTW distance.

Which of the lower bounds on the skyline should we use? Our insight is that we should use all of them in a cascade. We first use the $O(1)$ LB_KimFL, which while a very weak lower bound prunes many objects. If a candidate is not pruned at this stage we compute the $LB\_KeoghEQ$. Note that as discussed in Sections 4.1.3, 4.2.1, and 4.2.2, we can incrementally compute this; thus, we may be able to abandon anywhere between $O(1)$ and $O(n)$ time. If we complete this lower bound without exceeding the *best-so-far*, we reverse the query/data role and compute $LB\_KeoghEC$ (Section 4.2.3). If this bound does not allow us to prune, we then start the early abandoning calculation of DTW (Section 4.1.4).

Space limits preclude detailed analysis of which lower bounds prune how many candidates. Moreover, the ratios depend on the query, data, and size of the dataset. However, we note the following. Detailed analysis is available at Supporting Website[2]; lesion studies tell us that all bounds do contribute to speedup; removing any lower bound makes the search at least twice as slow; and finally, using this technique we can prune more than 99.9999% of DTW calculations for a large-scale search.

5. EXPERIMENTAL RESULTS

We begin by noting that we have taken extraordinary measures to ensure our experiments are reproducible. In particular, all data and code will be available in perpetuity, archived at Supporting Website[2]. Moreover, the site contains several videos that visualize some of the experiments in real time. We consider the following methods.

- *Naïve*. Each subsequence is Z-normalized from scratch. The full Euclidean distance or the DTW is used at each step. Approximately 2/3 of the papers in the literature do (some minor variant of) this.
- *State-of-the-art (SOTA)*. Each sequence is Z-normalized from scratch, early abandoning is used, and the LB_Keogh lower bound is used for DTW. Approximately 1/3 of the papers in the literature do (some minor variant of) this.
- *UCR Suite*. We use all of our applicable speedup techniques.

  DTW uses $R = 5\%$ unless otherwise noted. For experiments where Naïve or SOTA takes more than 24 hours to finish, we terminate the experiments and present the linearly extrapolated values, shown in gray. Where appropriate, we also compare to an oracle algorithm.
- *GOd's ALgorithm* (GOAL) is an algorithm that only maintains the mean and standard deviation using the online $O(1)$ incremental calculations.

It is easy to see that, short of an algorithm that precomputes and stores a massive amount of data (quadratic in m), GOAL is a lower bound on the fastest possible algorithm for either ED or DTW subsequence search with unconstrained and unknown length queries. The acronym reminds us that we would like to be as close to this goal value as possible.

[2] www.cs.ucr.edu/~eamonn/UCRsuite.html

It is critical to note that our implementations of Naïve, SOTA, and GOAL are efficient and tightly optimized, and they are not crippled in any way. For example, had we wanted to claim spurious speedup, we could implement SOTA recursively rather than iteratively, and that would make SOTA at least an order of magnitude slower. In particular, the code for Naïve, SOTA, and GOAL is exactly the same code as the UCR suite, except the relevant speedup techniques have been commented out.

While very detailed spreadsheets of all of our results are archived in perpetuity at Supporting Website[2], we present subsets of our results in the following. We consider wall clock time on a 2 Intel Xeon Quad-Core E5620 2.40GHz with 12GB 1333MHz DDR3 ECC Unbuffered RAM (using just one core unless otherwise explicitly stated).

5.1. Baseline Tests on Random Walk

We begin with experiments on random walk data. Random walks model financial data very well and are often used to test similarity search schemes. More importantly for us, they allow us to do reproducible experiments on massive datasets without the need to ship large hard drives to interested parties. We have simply archived the random number generator and the seeds used. We have made sure to use a very high-quality random number generator that has a period longer than the longest dataset we consider. In Table II, we show the length of time it takes to search increasingly large datasets with queries of length 128. The numbers are averaged over 1000, 100, and 10 queries, respectively.

These results show a significant difference between SOTA and the UCR suite. However, this is for a very short query; what happens if we consider longer queries? As we show in Figure 12, the ratio of SOTA-DTW over UCR-DTW improves for longer queries.

To reduce visual clutter we have only placed one Euclidean distance value on the figure, for queries of length 4096. Remarkably, UCR-DTW is even faster than SOTA *Euclidean* distance. As we shall see in our EEG and DNA examples, even though 4096 is longer than any published query lengths in the literature, there is a need for even longer queries.

It is also interesting to consider the results of the 128-length DTW queries as a ratio over GOAL. Recall that the cost for GOAL is independent of query length, and this experiment is just 23.57 seconds. The ratios for Naïve, SOTA, and the UCR suite are 5.27, 2.74, and 1.41, respectively. This suggests that we are asymptomatically closing in on the fastest possible subsequence search algorithm for DTW. Another interesting ratio to consider is the time for UCR-DTW over UCR-ED, which is just 1.18. Thus, the time for DTW is not significantly different than that for ED, an idea that contradicts an assumption made by almost all papers on time series in the last decade (including papers by the current authors).

5.2. Supporting Long Queries: EEG

The previous section shows that we gain the greatest speedup for long queries, and here we show that such long queries are really needed. The first user of the UCR suite was Dr. Sydney Cash, who, together with author Bilson Campana, wants to search massive archives of EEG data for examples of epileptic spikes, as shown in Figure 13.

From a single patient, Cash gathered 0.3 trillion datapoints and asked us to search for a prototypical epileptic spike Q he created by averaging spikes from other patients. The query length was 7000 points (0.23 seconds). Table III shows the results.

This data took multiple sessions over seven days to collect, at a cost of approximately $34,000 Supporting Website[2], so the few hours of CPU time we required to search the data are dwarfed in comparison. We refer the interested reader to Chaovalitwongse et al. [2005] and the references therein for a detailed discussion of the utility of similarity search in EEG data.

5.3. Supporting Very Long Queries: DNA

Most work on time series similarity search (and all work on time series indexing) has focused on relatively short queries, less than or equal to 1024 datapoints in length. Here we show that we can efficiently support queries that are two orders of magnitude longer.

We consider experiments with DNA that has been converted to time series. However, it is important to note that we are not claiming any particular bioinformatics utility for our work; it is simply the case that DNA data is massive, and the ground truth can be obtained through other means. As in Shieh and Keogh [2008], we use the algorithm in Table IV to convert DNA to time series.[3]

We chose a section of Human chromosome 2 (H2) to experiment with. We took a subsequence beginning at 5,709,500 and found its nearest neighbor in the genomes of five other primates, clustering the six sequences with single linkage to produce the dendrogram shown in Figure 14.

Pleasingly, the clustering is the correct grouping for these primates [Locke et al. 2011]. Moreover, because Human chromosome 2 is widely accepted to be a result of an end-to-end fusion of two progenitor ancestral chromosomes 2 and 3 [Locke et al. 2011], we should expect that the nearest neighbors for the nonhuman apes come from one of these two chromosomes, and that is exactly what we found.

Our query is of length 72,500, and the genome chimp is 2,900,629,179 base pairs in length. The single-core nearest neighbor search in the entire chimp genome took 38.7 days using Naïve, 34.6 days using SOTA, but only 14.6 hours using the UCR suite. As impressive as this is, as we shall show in the next section, we can do even better.

5.3.1. Can We Do Better Than the UCR Suite?—

We claim that for the problem of exact similarity search with arbitrary length queries, our UCR suite is close to optimal. However, it is instructive to consider an apparent counterexample and its simple patch.

Consider the search for a query of length 64 as considered in Section 5.1. Using GOAL took 9.18 seconds, but the UCR suite took only a little longer, just 10.64 seconds. Assume that the original query was:

[3]To preserve the reversible one-to-one mapping between time series and DNA we normalize the offset by subtracting *round*(*mean*) and we do not divide by the STD.

$$Q = [2.34,\ 2.01,\ 1.99, \ldots].$$

But we make it three times longer by padding it like this:

$$QP = [2.34,\ 2.34,\ 2.34,\ 2.01,\ 2.01,\ 2.01,\ 1.99,\ 1.99,\ 1.99, \ldots].$$

Further assume that we do the same to database *T*, to get *TP*, which is three times longer. What can we now say about the time taken for the algorithms? GOAL will take exactly three times longer, and Naïve takes exactly nine times longer, because each ED calculation takes three times longer and there are three times as many calculations to do. Our UCR suite does not take nine times longer, as it can partly exploit the smoothness of the data; however, its overhead is greater than three. Clearly, if we had known that the data was contrived in this manner, we could have simply made a one-in-three downsampled version of the data and query, done the search on this data, and reported the location and distance back in the *TP* space by multiplying each by three.

Of course, this type of pathologically contrived data does not occur in nature. However, some datasets are richly oversampled, and this has a very similar effect. For example, a decade ago, most ECGs were sampled at 256 Hz, and that seemed to be adequate for virtually all data analysis applications [Bragge et al. 2004]. However, current machines typically sample at 2048 Hz which, given this reasoning, would take up to sixty-four times longer to search ($(2048/256)^2$) with almost certainly identical results.

We believe that oversampled data can be searched more quickly by exploiting a provisional search in a downsampled version of the data that can quickly provide a low *best-so-far*, which, when projected back into the original space, can be used to prime the search by setting a low *best-so-far* at the beginning of the search, thus allowing the early abandoning techniques to be more efficient.

To test this idea, we repeated the experiment in the previous section, with a onein-ten downsampled version of the chimp genome/human query. The search took just 475 seconds. We denoted the best matching subsequence distance *rD*. We reran the full resolution search after initializing the *best-so-far* to *rD**10. This time the search fell from 14.64 hours to 4.17 hours, and we found the same answer, as we logically must.

Similar ideas have been proposed under the name of Iterative Deepening DTW [Adams et al. 2005] or Multi Scale DTW [Muller 2009; Zinke and Mayer 2006]; thus, we will not further develop this idea here. We simply caution the reader that oversampled (smooth) data may allow more speedup than a direct application of the UCR suite may initially suggest.

5.4. Realtime Medical and Gesture Data

The proliferation of inexpensive low-powered sensors has produced an explosion of interest in monitoring real time streams of medical telemetry and/or Body Area Network (BAN) data [Laerhoven et al. 2009].

There are dozens of research efforts in this domain that explicitly state that while monitoring under DTW is desirable, it is impossible [Wobbrock et al. 2007]. Thus, approximations of, or alternatives to DTW are used. Dozens of suggested workarounds have been suggested. For example, [Hsiao et al. 2005] resorts to only "dealing with shorter test and class templates, as this is more efficient," many research efforts including Stiefmeier et al. [2007] resort to a low cardinality version of DTW using integers, or DTW approximations that operate on piecewise linear approximations of the signals [Keogh et al. 2009; Pressly 2008], or drastically downsampled versions of the data [Gillian et al. 2011; Raghavendra et al. 2011]. In spite of some progress from existing ideas such as lower bounding, Alon et al. [2009] bemoans DTW is "still too slow for gesture recognition systems," Pressly [2008] laments that the "problem of searching with DTW (is) intractable," Gillian et al. [2011] says "Clearly (DTW) is unusable for real-time recognition purposes" and Srikanthan et al. [2011] notes "Processing of one hour of speech using DTW takes a few hours."

We believe that the UCR suite makes all of these objections moot. DTW *can* be used to spot gestures/brainwaves/musical patterns/anomalous heartbeats in real time, even on low-powered devices, even with multiple channels of data, and even with multiple simultaneous queries.

To see this, we created a dataset of one year of electrocardiograms (ECGs) sampled at 256Hz. We created this data by concatenating the ECGs of more than two hundred people, and thus we have a highly diverse dataset, with 8,518,554,188 datapoints. We created a query by asking USC cardiologist Dr. Helga Van Herle to produce a query she searches for on a regular basis, she created an idealized Premature Ventricular Contraction (PVC). The results are shown in Table V. While this was on our multi-core desktop machine, the fact that our results are 29,219 times faster than real-time (256 Hz) suggests that real-time DTW is tenable even on low-power devices.

5.5. Speeding Up Existing Mining Algorithms

In this section, we demonstrate that we can speed up much of the code in the time series data mining literature with minimal effort, simply by replacing their distance calculation subroutines with the UCR suite. In many cases, the difference is small, because the algorithms in question already typically try to prune as many distance calculations as possible. As an aside, in at least some cases we believe that the authors could benefit from redesigning the code in light of the drastically reduced cost for similarity search that the UCR suite offers. Nevertheless, even though the speedups are relatively small (1.5X to 16X), they are free, requiring just minutes of cut-and-paste code editing.

- *Time Series Shapelets* have garnered significant interest since their introduction in 2009 [Ye and Keogh 2009]. We obtained the original code and tested it on the Face (four) dataset, finding it took 18.9 minutes to finish. After replacing the similarity search routine with the UCR suite, it took 12.5 minutes to finish.
- *Online Time Series Motifs* generalize the idea of mining repeated patterns in a batch time series to the streaming case [Mueen and Keogh 2010]. We obtained the original code and tested it on the EEG dataset used in the original paper. The

fastest running time for the code, assuming linear space, is 436 seconds. After replacing the distance function with the UCR suite, it took just 156 seconds.

- *Classification of Historical Musical Scores* [Fornés et al. 2007]. This dataset has 4027 images of musical notes converted to time series. We used the UCR suite to compute the rotation-invariant DTW leave-one-out classification. It took 720.6 minutes. SOTA takes 142.4 hours. Thus, we have a speedup factor of 11.8.
- *Classification of Ancient Coins* [Huber-Mörk et al. 2011]. 2400 irregularly shaped coins are converted to time series of length 256, and rotation-invariant DTW is used to search the database, taking 12.8 seconds per query. Using the UCR suite, this takes 0.8 seconds per query.
- *Clustering of Star Light Curves* is an important problem in astronomy [Keogh et al. 2009], as it can be a preprocessing step in outlier detection [Rebbapragada et al. 2009]. We consider a dataset with 1000 (purportedly) phase-aligned light curves of length 1024, whose class has been determined by an expert [Rebbapragada et al. 2009]. Doing spectral clustering on this data with DTW (R =5%) takes about 23 minutes for all algorithms, and averaged over 100 runs we find the Rand-Index is 0.62. While this time may seem slow, recall that we must do 499,500 DTW calculations with relatively long sequences. As we do not trust the original claim of phase alignment, we further do rotation-invariant DTW that dramatically increases the Rand-Index to 0.76. Using SOTA, this takes 16.57 days, but if we use the UCR suite, this time falls by an order of magnitude, to just 1.47 days on a single core.

5.6. Optimal Ordering in Early Abandoning of Euclidean Distance

In this section we revisit the idea of early abandoning discussed in Section 4.2.2. We will show that if we order the distance calculations for a given time series query according to the absolute values of the Z-normalized values, then on average the cumulative square distance will be maximized. In other words, it is the best ordering in general, or in the average case.

THEOREM 1. *Given a time series, the position that gives the maximum average point-to-point distance is the position that has the highest absolute Z-normalized value*.

PROOF. After Z-normalization, the mean and standard derivation of a time series are 0 and 1, respectively.

For any fixed point a, the expected distance between this point and other points is

$$E\left[(X-a)^2\right] = E\left[X^2 - 2Xa + a^2\right] = E\left[X^2\right] - 2aE[X] + a^2E[1] = E\left[(X-\mu)^2\right] - 2a\mu + a^2 = Var(X) - 2a\mu + a^2 = 1 + a^2.$$

Hence, the expected point-to-point distance from a fixed point a to other points is $1+a^2$ Therefore, the point that has the maximum absolute Z-normalized value will maximize the expected point-to-point distance. □

LEMMA 1. *The optimal ordering for early abandoning of Euclidean distance is ordering by the absolute Z-normalized value*.

PROOF. The Euclidean distance is a summation of all point-to-point distances and it is a monotonically nondecreasing function. From Theorem 1, if we order the points in the query by their absolute of Z-normalized values of each point, the expected contribution to the summation will be increased as much as possible. Hence, the number of points used for early abandoning is minimized, and this is the optimal ordering. □

According to Lemma 1, we can obtain the optimal ordering in early abandoning simply by ordering by absolute Z-normalized value. Next, we will empirically consider the number of point-to-point calculations needed in the early abandoning process. The UCR suite computes a smaller number of calculations in finding the nearest neighbor compared to SOTA, Figure 15 shows the typical behavior of the number of point-to-point calculations in the early abandoning process from SOTA and the UCR suite when finding the nearest neighbor of a query of size 128 in a random walk time series of size 100 million.

As we expect, the average number of point-to-point calculations is greatly reduced when a very good candidate (a small *best-so-far*) has been found. Using the traditional Naïve approach, exactly 128 calculations per datapoint are required if one does not apply the early abandon technique. With the ordering, the UCR suite can reduce the average number of calculations to 3.34, or just 2.6% of the query length. When the query is ten times longer, of length 1280, the average number of calculated point-to-point distances from our UCR suite is only 11.49, or less than 1% of the length of the query.

When the query is longer, the number of point-to-point calculations is obviously higher; however, if we consider the ratio between the number of calculations and the query length, surprisingly, this ratio typically decreases for longer queries. Figure 16 shows the relationship between the average number of point-to-point calculations and the length of the query on a 100-million random walk time series (see also Figure 15).

According to Lemma 1, it is possible to order by the values of either the query or the incoming data. The advantage in ordering the query is that we can see the query ahead of time and we can do the ordering just once and use that ordering for the entire search process.

In summary, in this section we have explained how to order the individual elements of the distance calculations in order to maximize the utility of early abandoning in Euclidean distance, and we show that a simple solution of ordering according to its Z-normalized values is optimal. However, generalizing this solution to early abandoning with LB_Keogh is not easy because the contribution of each point is dependent upon the shape of the envelope. Empirically, however, it does seem that the solution used for Euclidean distance also works very well for LB_Keogh, although a proof is elusive.

6. SUPPORTING THE UNIFORM SCALING DISTANCE

In this section we show that most of the ideas we have introduced to speed up ED and DTW also work for the *uniform scaling* (US) distance. While DTW is a generalization of ED to

allow invariance to local warping or changes of scale, US is a generalization of ED to allow global invariance to scale. While the need for scaling invariance shows up in many domains, it is easiest to understand if we consider examples in speech or music. For ease of exposition we consider the analogue examples in ASCII text; later we will show visual examples directly in the time series space.

Consider the opening phrase of Shakespeare's most famous soliloquy, with spaces indicating the length of pauses:

To be or not to be

If a naive speaker reads this, he may pronounce it as:

To be or not to be

The hamming distance (the discrete analogue of ED) would find these utterances very different, since they disagree in all of the underlined locations. However, the string edit distance (the discrete analogue of DTW) would find them essentially identical,[4] since they only differ locally.

Let us imagine now that the speaker pronounces the line very slowly, for overly dramatic effect:

TToo bbee oorr nnoott ttoo bbee

Here, even the string edit distance would report a huge difference between this utterance and the original, as they disagree in all the underlined locations. This is in spite of the fact that a listener (or reader) would easily recognize the intended sentence. The solution is to rescale this drawn-out version, by uniformly scaling it (downsampling) by a factor of two.

This is what the uniform scaling distance does in the real-valued domain. The task is more difficult in the real-valued domain, as the scaling factor does not have to be an integer (we can rescale the ASCII text bbee by a factor of two to get be but we cannot scale it by a factor of three or two point seven).

In the next section, we show that we can use some of the ideas introduced in this work to support uniform scaling. Please note that US and DTW are not mutually exclusive; they can be used in conjunction [Fu et al. 2008]. However, for clarity of presentation, we consider them separately in this work.

6.1. Why Uniform Scaling?

Although nearest neighbor search with the Euclidean distance is efficient, the quality/correctness of the neighbors may be dubious for some real-world problems. As empirical studies [Keogh et al. 2004; Vlachos et al. 2003] suggest, both ED and DTW may not be

[4]In some flavors of string edit distance, the cost of inserting/deleting white space is zero.

suitable (or at least, fully sufficient) for certain problems. Figure 17 demonstrates a synthetic example where uniform scaling distance can outperform both ED and DTW. While we formally define uniform scaling in the following, it suffices to say here that uniform scaling refers to the global linear stretching/shrinking of patterns (in contrast to warping, which refers to local and nonlinear stretching/shrinking). In our toy example, we generated a simple set of time series and then created a single-linkage clustering of them. Neither ED nor DTW can correctly cluster these simple shapes, in spite of the fact that they are easily clustered by humans. Here, uniform scaling can discover the correct clustering as shown in Figure 17(right). Note that DTW and uniform scaling each have a single parameter to set; the size of warping windows for DTW and the maximum scaling factor for uniform scaling; however, we manually set the best parameter for both approaches.

Although the shapes of the time series are simple, the number of peaks in the time series is different. The local alignment ability of DTW can align the different positions of peaks but it is not robust to different numbers of peaks, as the example in Figure 17 shows. In contrast, uniform scaling has been shown to be useful in many situations [Keogh et al. 2004; Vlachos et al. 2003] because it can rescale one subsequence to match another in the correct scale, truncating off spurious data in one sequence if necessary. While this synthetic example is somewhat contrived for visual clarity, existing work has shown that uniform scaling invariance can be very useful for real datasets [Keogh et al. 2004]. Moreover, in Section 6.5 we will show an example of a real-world problem in speech processing, where only uniform scaling can produce correct answers. In the next section, we will explain how to compute the uniform scaling distance.

6.2. How to Compute Uniform Scaling

As noted in the previous section, uniform scaling is a technique to relax an assumption about the correct length of the answer subsequence; this technique can be used to either remove a parameter, *fixed length*, or limit the range or *expected length* of the answer. A subsequence can be rescaled into another length called a *scaling subsequence*.

Definition 4. A *scaling subsequence* of length s from subsequence Q of length n is a subsequence of length s ($s \leq n$) mapped linearly from Q. Precisely, given a subsequence = $q_1, q_2, \ldots, q_n$, a scaling subsequence $\hat{Q}^s$ of length s is defined as $\hat{Q}^s = \hat{q}_1, \hat{q}_2, \ldots, \hat{q}_s$, where an element $\hat{Q}_i^s$ or $\hat{q}_i = q_{\lceil i * n/s \rceil}$, $1 \leq i \leq s$. A set of all scaling subsequences is defined as $\hat{Q} = \{\hat{Q}^s | 1 \leq s \leq n\}$.

Note that every scaling subsequence $\hat{Q}^s$ is a Euclidean Z-normalized subsequence. The distance between two subsequences has been defined in Definition 3 (Section 3.1). We can define the distance between two subsequences of different lengths.

Definition 5. The Euclidean distance (ED) between a scaling subsequence $\hat{Q}^s$ and C is defined as:

$$ED(\hat{Q}^s, C) = \sqrt{\sum_{i=1}^{s} (\hat{q}_i - c_i)^2}.$$

The minimum distance among all scaling subsequences and a candidate subsequence is called the *uniform scaling distance*.

Definition 6. The *uniform scaling Euclidean distance* (or just *uniform scaling distance*) between a set of scaling subsequences $\hat{Q}$ and another subsequence C is defined as:

$$Uniform\_scaling\_Dist(\hat{Q}, C) = \min_s ED(\hat{Q}^s, C).$$

An example of a scaling subsequence is shown in Figure 18(left). The original subsequence Q of length 100 is shown on the top, and after rescaling by Definition 4, the subsequence Q of length 77, or $\hat{Q}^{77}$, is shown as the bottom subsequence. The value in the scaling subsequence is linearly mapped from the value of the original subsequence.

The distance between the scaling subsequence and a candidate is shown in Figure 18(right). The top figure shows the distance between $\hat{Q}^{100}$, or the original subsequence, and the candidate C. The bottom figure shows $ED(\hat{Q}^{77}, C)$ as the summation of the square of the hatch lines. In this example, after the correct scaling, much the distance becomes smaller than the original Euclidean distance.

To compute the uniform scaling distance, the simple goal is to find the candidate whose uniform scaling distance is minimized. One possible approach is using the fastest known algorithm (the UCR suite) to search the nearest neighbor of every possible length and pick the smallest distance among all scales. However, even though the UCR suite is very fast, running it, say, 50 times to find the best match to a query of length 100 to any candidate match in the range 75 to 125 would clearly introduce a massive overhead. Fortunately, as we show in the next section, the UCR suite can exactly handle uniform scaling with very little overhead.

6.3. UCR Suite: Uniform Scaling

To handle uniform scaling directly with the UCR suite, we use the idea of grouping all scaling subsequences together as proposed by Keogh et al. [2004]. The idea is simple but very effective. We begin by grouping all scaling subsequences together and create representatives called *upper envelope* and *lower envelope*.

Definition 7. The *upper envelope* U and *lower envelope* L of a set of scaling subsequences $\hat{Q}$ is defined as:

$$U = u_1, u_2, \ldots, u_s \text{ where } u_i = \max(\hat{Q}_i^1, \hat{Q}_i^2, \ldots, \hat{Q}_i^n)$$
$$L = l_1, l_2, \ldots, l_s \text{ where } l_i = \min(\hat{Q}_i^1, \hat{Q}_i^2, \ldots, \hat{Q}_i^n).$$

where s is the user-defined minimum length of scaling subsequence $(s \leq n)$. We also define a scaling factor by $1 - s/n$. If the scaling factor is 0%, the uniform scaling search degenerates to a Euclidean distance search.

Recall that the set of all scaling subsequences $\hat{Q}$ contains scaling subsequence $\hat{Q}^k$ of any length $k(s \leq k \leq n)$, and $\hat{Q}_i^k$ is the ith value of the scaling subsequence $\hat{Q}^k$. Hence, the upper/lower envelope is a sequence of all maximum/minimum values of all scaling subsequences. To demonstrate this, a visual example of the upper/lower envelopes is shown in Figure 19. Figure 19(left) shows all scaling subsequences of different lengths created from the subsequence Q. The envelopes are created from all values inside those scaling subsequences; the upper envelope U is composed of all maximum values and the lower envelope L is composed of all minimum values from those subsequences. The length of the envelopes is equal to the length of the shortest scaling subsequence.

After the envelopes are created, we can use LB_Keogh to speed up the nearest neighbor search, in this case *uniform scaling search*. Figure 19(top-right) and (bottom) shows examples of the upper and lower envelopes when the length is set to 80 and 60, respectively. LB_Keogh can be visualized as the summation of the hatch lines in Figure 19(right). Although the envelopes are not as long as the query, LB_Keogh can still provide a lower bound and can be used to prune candidates efficiently. Note that the early abandoning technique and other techniques from the UCR suite (Sections 4.1 and 4.2) can also be applied here.

An overview of the uniform scaling search algorithm is shown in Table IV. Note that all techniques in the UCR suite, e.g., online Z-normalization, early abandoning, and so on, can be applied here, however for the sake of clarity, these techniques are omitted.

To perform uniform scaling search, one parameter is required from a user: the maximum scaling factor. Without loss of generality, we use the scaling factor to scale down the size of the query. Please note that this parameter can be used to control the range of the final answer and it is not that hard to set this parameter if the query length is known; however, it is also possible to hide this parameter from end-users by allowing hardcoded scaling factors of, say, up to 200% /down to 50%.

The algorithm starts by setting the *best-so-far* value to the maximum possible value (line 1). Next, the algorithm reads the data once with the online Z-normalization technique (Table I); the current candidate from the data is stored in C (line 4). Then, in line 6, the envelopes are generated, whose size depends on the value of the scaling factor. Classic LB_Keogh is calculated according to the envelopes (line 7) and if the lower bound distance is smaller than the current *best-so-far*, the distances from all scaling subsequences to the current candidate will be computed (lines 9–11). Hence, the best answer will be selected as the nearest neighbor in the end.

6.4. Experimental Results: Scalability

We compare our uniform scaling search algorithm, UCR-US, in Table III with the Naïve approach, which is to search a nearest neighbor using the best known algorithm, the UCR suite, many times, and select the best neighbor among different lengths.

Figure 20 shows that, while producing the exact same result, the proposed approach, UCR-US, is faster than the Naïve approach by an order of magnitude. Again, we emphasize that the Naïve approach uses UCR-ED as the search algorithm, and thus all of the speedup is directly attributable to the uniform scaling lower bound. When the query is longer, our approach is much faster. The running time in this experiment is an averaged running time from ten random walk queries of various lengths (32 to 2048) on a random walk time series of length 100 million datapoints using a single core machine with a scaling factor of 10%.

When the scaling factor is increased, the speedup gained by using our algorithm is also significantly increased. Figure 21 shows experiments on searching uniform scaling nearest neighbor on random walk data of size 100 million, with the query length fixed at 128. The number of candidate pairs grows dramatically if the range of possible lengths is increased. However, using our approach, grouping all scaling subsequences to envelopes can reduce the number of needed calculations by two orders of magnitude.

THEOREM 2. *LB_Keogh is a lower bound of all of the uniform scaling Euclidean distances*.

PROOF. Refer to Keogh et al. [2004]. □

6.5. On the Utility of Uniform Scaling Search: The Raven

In this section, we will demonstrate the utility of uniform scaling search. In particular, we show that it can produce superior results compared to ED/DTW search, especially if the query is long. We consider the audio of an actor reading a well-known poem, *The Raven*, by Edgar Allan Poe. We convert the audio into a single time series using Mel Frequency Cepstral Coefficients (MFCC).

We did a search for a long phrase that appears once in the poem. That phrase is “tapping, tapping at my chamber door.” The results of three nearest neighbor search algorithms are shown in Figure 22. Figure 22(top) shows the time series created from the audio file with the position of the query and the results. Each algorithm finds the nearest neighbor in the time series space and projects the results back to the audio file; we can visualize the corresponding text in Figure 22(bottom). Although the phrase “tapping, tapping at my chamber door” appears only once in the poem, the uniform scaling search objectively outperforms other search algorithms (as measured by, say, string edit distance in the ASCII space).

7. DISCUSSION AND CONCLUSIONS

While our work has focused on fast sequential search, we believe that for both DTW and US, our work is faster than all known indexing efforts. Consider Assent et al. [2008], which indexes a random walk time series of length 250,000 to support queries of length 256. They

built various indexes to support DTW queries, noting that the fastest of the four carefully-tuned approaches requires access to approximately 15,000 pages to answer a query. These disk accesses are necessarily random accesses. While they did not give wall clock time, if we assume an HDD spindle speed of 7200 rpm (average rotational latency = 4.17ms), then just the disk I/O time to answer this query must be at least 62.55 seconds. However, as we have shown, we can load all of the data into the main memory with more efficient sequential disk accesses and answer these queries in 0.4 seconds, including disk I/O time, on a single core machine.

Note that all experiments in this article include the time taken to read the data from disk. However, for more than a few million objects this time is inconsequential; thus, we did not report it separately. We have made a strong and unintuitive claim in the abstract. We said that our UCR-*DTW* is faster than all current Euclidean distance searches. In Table V, for example, we show that DTW can be three times faster than state-of-the-art ED searching. How is this possible? Recall that all Euclidean searches in the literature require an $O(n)$ data normalizing step to be performed for each subsequence. Thus, no matter how effective the pruning/search strategy used, the amortized time for a single sequence must be at least $O(n)$. In contrast, using the ideas developed in this work, the vast majority of potential DTW calculations are pruned with $O(1)$ work, while some require up to $O(n)$ work, and only a vanishingly small fraction require $O(nR)$ work. The weighted average of these possibilities is less than $O(n)$.

To put our results in perspective, we compare them with a very recent state-ofthe art embedding-based DTW search technique, called EBSM (including the variant called BSE) [Papapetrou et al. 2011]. This is an excellent paper to use as a benchmark, as it exhaustively compares to almost all other methods in the literature, and it tests its contributions over different datasets, query lengths, warping widths, and so on. The contrast between EBSM and our method are summarized as follows.

- Our method is exact; EBSM is approximate.
- EBSM requires setting some parameters (number of reference sequences, dimensionality, number of split points, and so on). Our method requires zero parameters.
- EBSM requires offline preprocessing that takes over 3 hours for just 1 million objects. We have zero preprocessing time.
- The EBSM method does not, and cannot, Z-normalize. As noted in Section 1.2.1, we believe that Z-normalizing is critical, and we have shown that failure to do it hurts on 45 out of 45 of the UCR time series classification datasets.
- EBSM can support queries in a limited range, which must be predetermined and limited for efficiency. In contrast, we have no minimum/maximum query length.
- We can handle exact queries under uniform scaling [Fu et al. 2008], whereas EBSM cannot do this.
- Finally, we are simply much faster! (Section 1).

Note, however, that there can be great utility in fast approximate search. There exist data mining algorithms that can use a combination of (hopefully few) exact distance measures and (hopefully much faster) approximate searches to produce overall exact results [Shieh and Keogh 2008]. However, an approximate search method being faster than our approach is a very high threshold to meet.

We have shown that our suite of ideas is 2 to 164 times faster than the true state-of-the-art, depending on the query/data. However, based on the quotes from papers that we have sprinkled throughout this work, we are sometimes more than 100,000 times faster than recent papers; how is this possible? The answer seems to be that it is possible to produce very Naïve implementations of DTW. For example, the recursive version of DTW can be one to three orders of magnitude slower than the iterative version, depending on the computer language and query length. Thus, the contributions of this article are twofold. First, we have shown that much of the recent pessimism about using DTW for real-time problems was simply unwarranted [Ding et al. 2008]. If carefully implemented, existing techniques, especially lower bounding, can make DTW tractable for many problems. Our second contribution is the introduction of the UCR suite of techniques that make DTW and Euclidean distance subsequence search significantly faster than current state-of-the-art techniques.

We have avoided presenting full pseudo-code to enhance the readability of the text; however, full pseudo-code (and highly useable source-code) *is* readily available at Supporting Website[2]. In future work we plan to revisit algorithms for time series motif discovery [Mueen and Keogh 2010; Mueen et al. 2011], anomaly detection [Rebbapragada et al. 2009; Shieh and Keogh 2008], time series summarization, shapelet extraction [Ye and Keogh 2009], clustering, and classification [Ding et al. 2008], in light of the results presented in this work.

ACKNOWLEDGMENTS

We thank all the donors of code and data. We thank the reviewers for their useful comments. Papapetrou et al. [2011] omits the length of the test data we mention in Section 1. T. Rakthanmanon was kind enough to give us the 1 million figures.

This work is supported by the National Science Foundation grants 0803410 and 0808770, FAPESP award 2009/06349–0, and the Royal Thai Government scholarship.

REFERENCES

Adams N, Marquez D, and Wakefield G 2005 Iterative deepening for melody alignment and retrieval. In Proceedings of ISMIR. 199–206.

Alon J, Athitsos V, Yuan Q, and Sclaroff S 2009 A unified framework for gesture recognition and spatiotemporal gesture segmentation. IEEE Trans. Pattern Anal. Mach. Intell 31, 9, 1685–1699. [PubMed: 19574627]

Assent I, Krieger R, Afschari F, and Seidl T 2008 The TS-Tree: Efficient time series search and retrieval. In Proceedings of EDBT. 252–63.

Bei CD and Gray RM 1985 An improvement of the minimum distortion encoding algorithm for vector quantization. IEEE Trans. Commun 33, 10, 1132–1133.

Bragge T, Tarvainen MP, and Karjalainen PA 2004 High-Resolution QRS Detection Algorithm for Sparsely Sampled ECG Recordings. Department of Applied Physics Report, University of Kuopio.

Chadwick NA, McMeekin DA, and Tan T 2011 Classifying eye and head movement artifacts in EEG Signals. In Proceedings of IEEE DEST. 285–291.

Chan TF, Golub GH, and Leveque RJ 1983 Algorithms for computing the sample variance: Analysis and recommendations. Amer. Statist 37, 242–247.

Chaovalitwongse WA, Sachdeo RC, Pardalos PM, Iasemidis LD, and Sackellares JC 2005 Automated brain activity classifier. Epilepsia 46, 313.

Chen L and Ng R 2004 On the marriage of LP-norms and edit distance. In Proceedings of VLDB. 792–803.

Chen Y, Chen G, Chen K, and Ooi BC 2009 Efficient processing of warping time series join of motion capture data. In Proceedings of ICDE. 1048–1059.

Cheng DY, Gersho A, Ramamurthi B, and Shoham Y 1984 Fast search algorithms for vector quantization and pattern matching. In Proceedings of ICASSP. 372–375.

Ding H, Trajcevski G, Scheuermann P, Wang X, and Keogh EJ 2008 Querying and mining of time series data: Experimental comparison of representations and distance measures. J. VLDB 1, 2, 1542–1552.

Dupasquier B and Burschka S 2011 Data mining for hackers–Encrypted traffic mining. In Proceedings of the 28th Chaos Comm' Congress.

Fornés A, Lladós J, and Sanchez G 2007 Old handwritten musical symbol classification by a dynamic time warping based method. Graph. Recogn 5046, 51–60.

Fu A, Keogh EJ, Lau L, Ratanamahatana C, and Wong R 2008 Scaling and time warping in time series querying. VLDB J. 17, 4, 899–921.

Gillian N, Knapp R, and O'Modhrain S 2011 Recognition of multivariate temporal musical gestures using n-dimensional dynamic time warping. In Proceedings of the 11th International Conference on New Interfaces for Musical Expression.

Goldberg D 1991 What every computer scientist should know about floating-point arithmetic. ACM Comput. Surv 23, 1.

Guitel G 1975 Histoire Comparée des Numérations Écrites. Flammarion, Paris 566–574.

Hsiao M, West K, and Vedatesh G 2005 Online context recognition in multisensor system using dynamic time warping. In Proceedings of ISSNIP. 283–288.

Huber-Mörk R, Zambanini S, Zaharieva M, and Kampel M 2011 Identification of ancient coins based on fusion of shape and local features. Mach. Vis. Appl 22, 6, 983–994.

Jegou H, Douze M, Schmid C, and Perez P 2010 Aggregating local descriptors into a compact image representation. In Proceedings of IEEE CVPR. 3304–3311.

Kahveci T and Singh AK 2004 Optimizing similarity search for arbitrary length time series queries. IEEE Trans. Knowl. Data Eng 16, 4, 418–433.

Keogh E and Ratanamahatana CA 2005 Exact indexing of dynamic time warping. Knowl. Inform. Syst 7, 3, 358–386.

Keogh EJ and Kasetty S 2003 On the need for time series data mining benchmarks: A survey and empirical demonstration. Data Mining Knowl. Discov 7, 4, 349–371.

Keogh EJ, Palpanas T, Zordan VB, Gunopulos D, and Cardle M 2004 Indexing large human-motion databases. In Proceedings of VLDB. 780–791.

Keogh EJ, Wei L, Xi X, Vlachos M, Lee SH, and Protopapas P 2009 Supporting exact indexing of arbitrarily rotated shapes and periodic time series under Euclidean and warping distance measures. VLDB J. 18, 3, 611–630.

Kim S, Park S, and Chu W 2001 An index-based approach for similarity search supporting time warping in large sequence databases. In Proceedings of ICDE. 607–614.

Laerhoven K, Berlin E, and Schiele B 2009 Enabling efficient time series analysis for wearable activity data. In Proceedings of ICMLA. 392–397.

Lim SH, Park H, and Kim SW 2007 Using multiple indexes for efficient subsequence matching in time-series databases. Inf. Sci 177, 24, 5691–5706.

Ling RF 1974 Comparison of several algorithms for computing sample means and variances. J. Amer. Statist. Assoc 69, 348, 859–866.

Locke DP, Hillier LW, Warren WC, et al. 2011 Comparative and demographic analysis of orangutan genomes. Nature 469, 529–533. [PubMed: 21270892]

Masek WJ and Paterson MS 1980 A faster algorithm computing string edit distances. J. Comput. Syst. Sci 20, 1, 18–31.

McNames J 2000 Rotated partial distance search for faster vector quantization encoding. IEEE Signal Proc. Lett 7, 9, 244–246.

Mueen A and Keogh EJ 2010 Online discovery and maintenance of time series motifs. In Proceedings of KDD. 1089–1098.

Mueen A, Keogh EJ, Zhu Q, Cash S, Westover MB, and Shamlo N 2011 A disk-aware algorithm for time series motif discovery. Data Min. Knowl. Discov 22, 1–2, 73–105.

Muller M 2009 Analysis and retrieval techniques for motion and music data. EUROGRAPHICS tutorial.

Papapetrou P, Athitsos V, Potamias M, Kollios G, and Gunopulos D 2011 Embedding-based subsequence matching in time-series databases. ACM Trans. Datab. Syst 36, 3, 174.

Pressly W 2008 TSPad: A Tablet-PC based application for annotation and collaboration on time series data. In Proceedings of ACM Southeast Regional Conference. 527–552.

Raghavendra B, Bera D, Bopardikar A, and Narayanan R 2011 Cardiac arrhythmia detection using dynamic time warping of ECG beats in e-healthcare systems. In Proceedings of WOWMOM. 1–6.

Rebbapragada U, Protopapas P, Brodley C, and Alcock C 2009 Finding anomalous periodic time series. Mach. Learn 74, 3, 281–313.

Sakurai Y, Yoshikawa M, and Faloutsos C 2005 FTW: Fast similarity search under the time warping distance. In Proceedings of PODS. 326–337.

Sakurai Y, Faloutsos C, and Yamamuro M 2007 Stream monitoring under the time warping distance. In Proceedings of ICDE. 1046–1055.

Shieh J and Keogh EJ 2008 *i*SAX: Indexing and mining terabyte sized time series. In Proceedings of KDD. 623–631.

Srikanthan S, Kumar A, and Gupta R 2011 Implementing the dynamic time warping algorithm in multithreaded environments for real time and unsupervised pattern discovery. In Proceedings of IEEE ICCCT. 394–398.

Stiefmeier T, Roggen D, and Tröster G 2007 Gestures are strings: Efficient online gesture spotting and classification using string matching. In Proceedings of the ICST 2nd International Conference on Body Area Networks.

Vlachos M, Hadjieleftheriou M, Gunopulos D, and Keogh EJ 2003 Indexing multi-dimensional time-series with support for multiple distance measures. In Proceedings of KDD. 216–225.

Whitney CR 1997 Jeanne Calment, World's elder, dies at 122. New York Times (8/5/97).

Wobbrock JO, Wilson AD, and Li Y 2007 Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In Proceedings of ACM UIST. 159–168.

Ye L and Keogh EJ 2009 Time series shapelets: A new primitive for data mining. In Proceedings of KDD. 947–956.

Yi B, Jagadish H, and Faloutsos C 1998 Efficient retrieval of similar time sequences under time warping. In Proceedings of ICDE. 201–208.

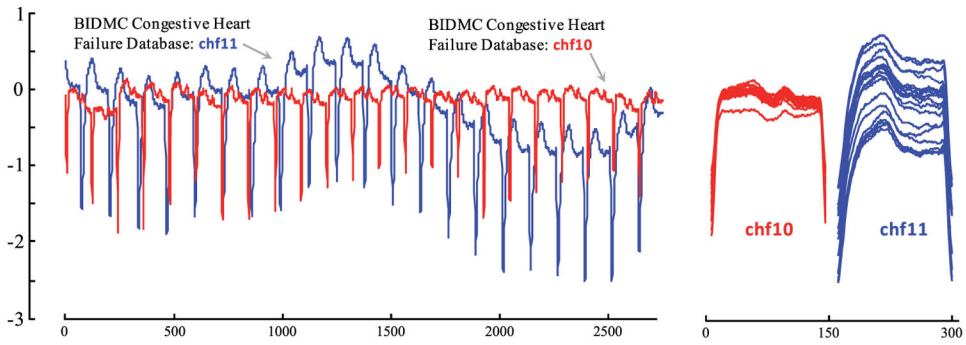


Fig. 1.

(left) Examples of approximately eleven seconds of ECG data from a 22-year-old male (chf10) and a 54-year-old female (chf11), both with severe congestive heart failure. Note that both traces, but especially chf11, exhibit wandering baseline. (right) Without any normalization for offset or amplitude, we extracted 21 and 20 full heartbeats, respectively.

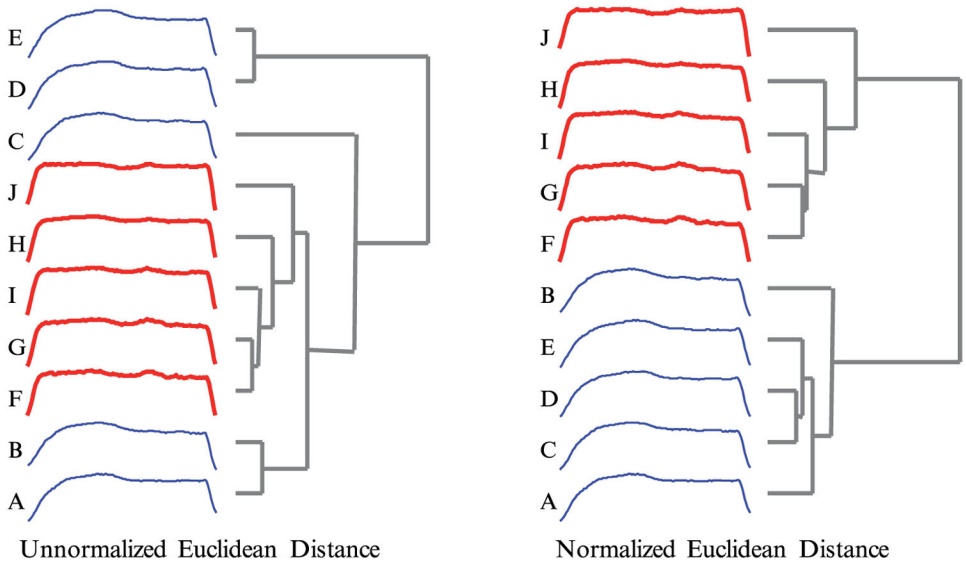


Fig. 2.

(left) A single-linkage hierarchical clustering of ten beats randomly chosen from those extracted in Figure 1, using the unnormalized Euclidean distance. (right) A single-linkage hierarchical clustering of the same ten beats using normalized Euclidean distance. Only about one quarter of the data was clustered for clarity; other sized subsets have similar outcomes. The randomly extracted heartbeats are: 1, 2, 8, 12, 14 and 2, 4, 9, 17, 19.

Fig. 3.

Screen captures from the original video from which the Gun/NoGun data was culled. The center frame is the original size; the left and right frames have been scaled by 110% and 90%, respectively. While these changes are barely perceptible, they double the error rate if normalization is not used. (Video courtesy of Dr. Ratanamahatana).

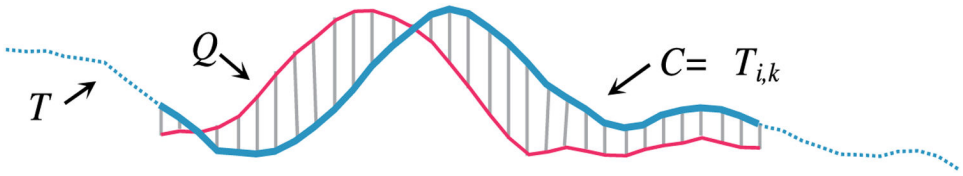


Fig. 4.

A long time series T can have a subsequence $T_{i,k}$ extracted and compared to a query Q under the Euclidean distance, which is simply the square root of the sum of the squared hatch line lengths.

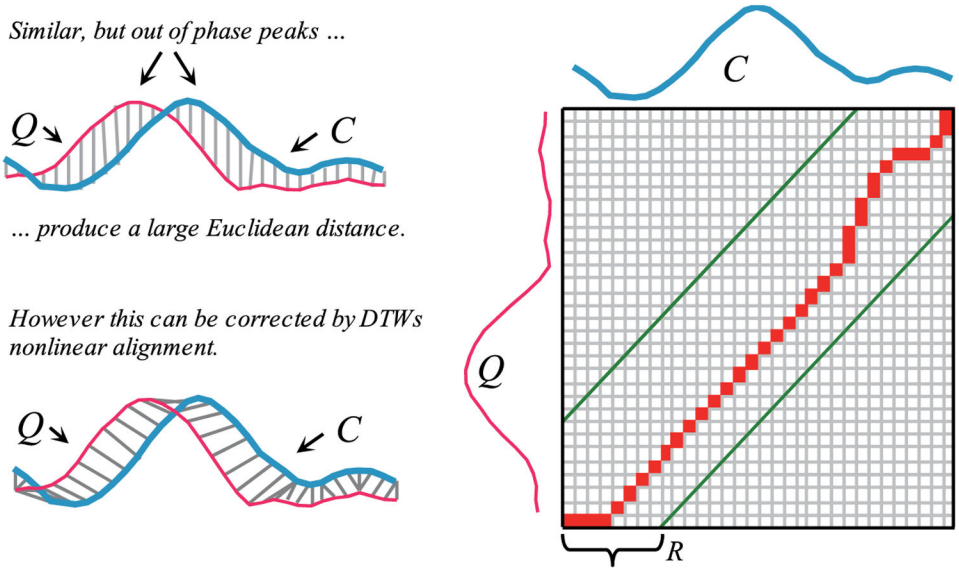


Fig. 5.

(left) Two time series that are similar but out of phase. (right) To align the sequences we construct a warping matrix, and search for the optimal warping path (red/solid squares). Note that Sakoe-Chiba Band that has width R is used to constrain the warping path.

Fig. 6.

(left) The LB_KimFL lower bound is $O(1)$ and uses the distances between the First (Last) pair of points from C and Q as a lower bound. It is a simplification of the original LB_Kim [Kim et al. 2001]. (right) The LB_Keogh lower bound is $O(n)$ and uses the Euclidean distance between the candidate sequence C and the closer of $\{U, L\}$ as a lower bound.

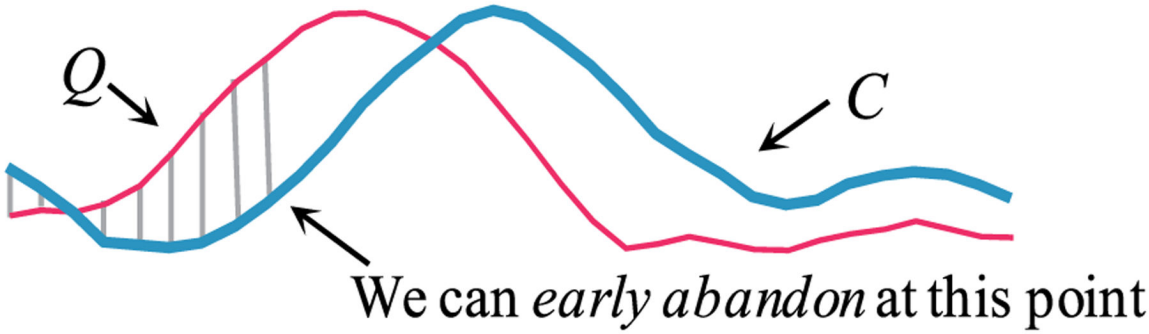


Fig. 7.

An illustration of ED *early abandoning*. We have a *best-so-far* value of *b*. After incrementally summing the first nine (of thirty-two) individual contributions to the ED we have exceeded *b*, thus it is pointless to continue the calculation [Keogh et al. 2009].

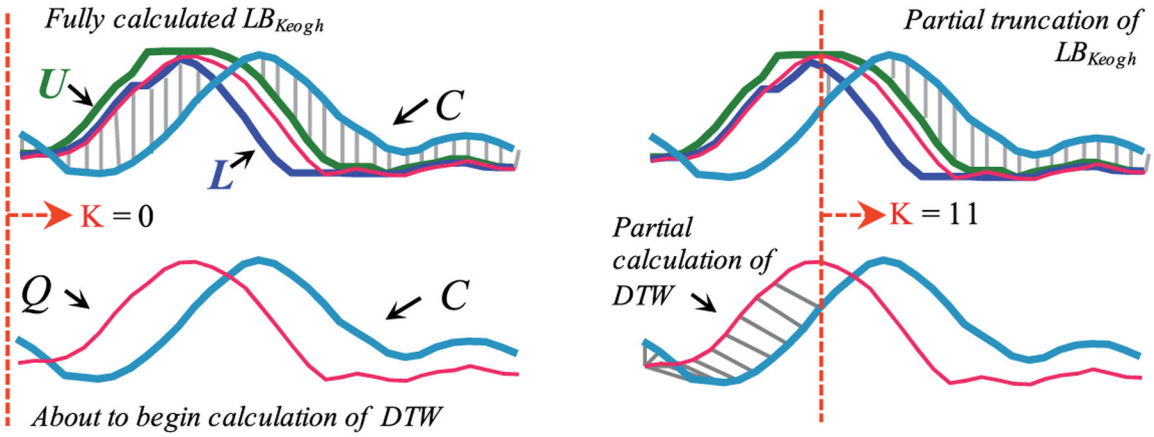


Fig. 8.

(left) At the top we see a completed LB_{Keogh} calculation, and below it we are about to begin a full DTW calculation. (right) We can imagine the orange/dashed line moving from left to right. If we sum the LB_{Keogh} contribution from the right of the dashed line (top) and the partial (incrementally calculated) DTW contribution from the left side of the dashed line (bottom), this is will be a lower bound to DTW(Q, C).

Fig. 9.

(left) ED *early abandoning*. We have a *best-so-far* value of *b*. After incrementally summing the first nine individual contributions to the ED, we have exceeded *b*; thus, we abandon the calculation. (right) A different ordering allows us to abandon after just five calculations.

Fig. 10.

(left) Normally the LB_Keogh envelope is built around the query (see also Figure 6(right), and the distance between C and the closer of $\{U, L\}$acts as a lower bound (right). However, we can reverse the roles such that the envelope is built around C and the distance between Q and the closer of $\{U, L\}$ is the lower bound.

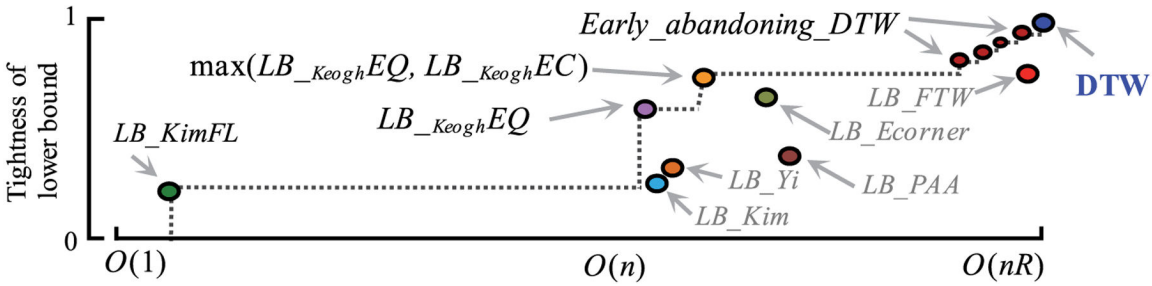


Fig. 11.

The mean tightness of selected lower bounds from the literature plotted against the time taken to compute them.

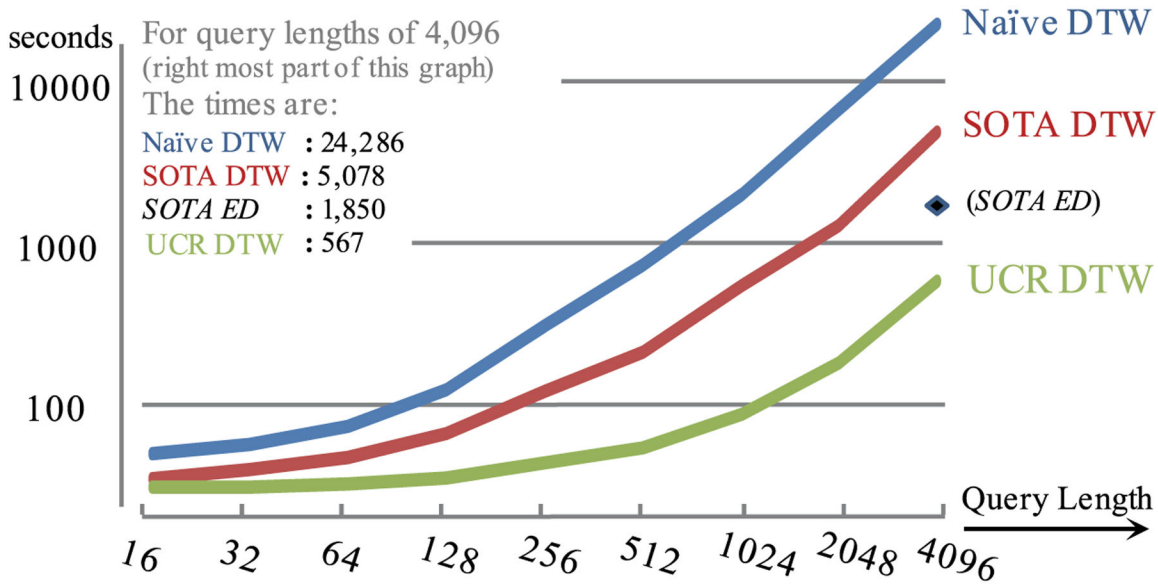


Fig. 12.

The time taken to search random walks of length 20 million with increasingly long queries, for three variants of DTW. In addition, we include just length 4096 with SOTA-ED for reference.

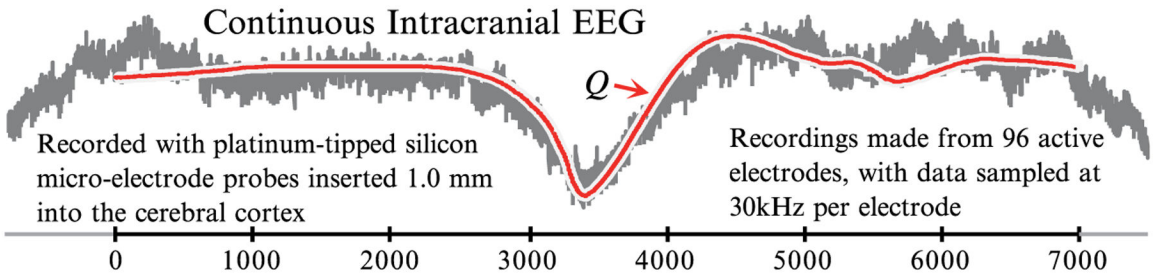


Fig. 13.
Query *Q* shown with a match from the 0.3 trillion EEG dataset.

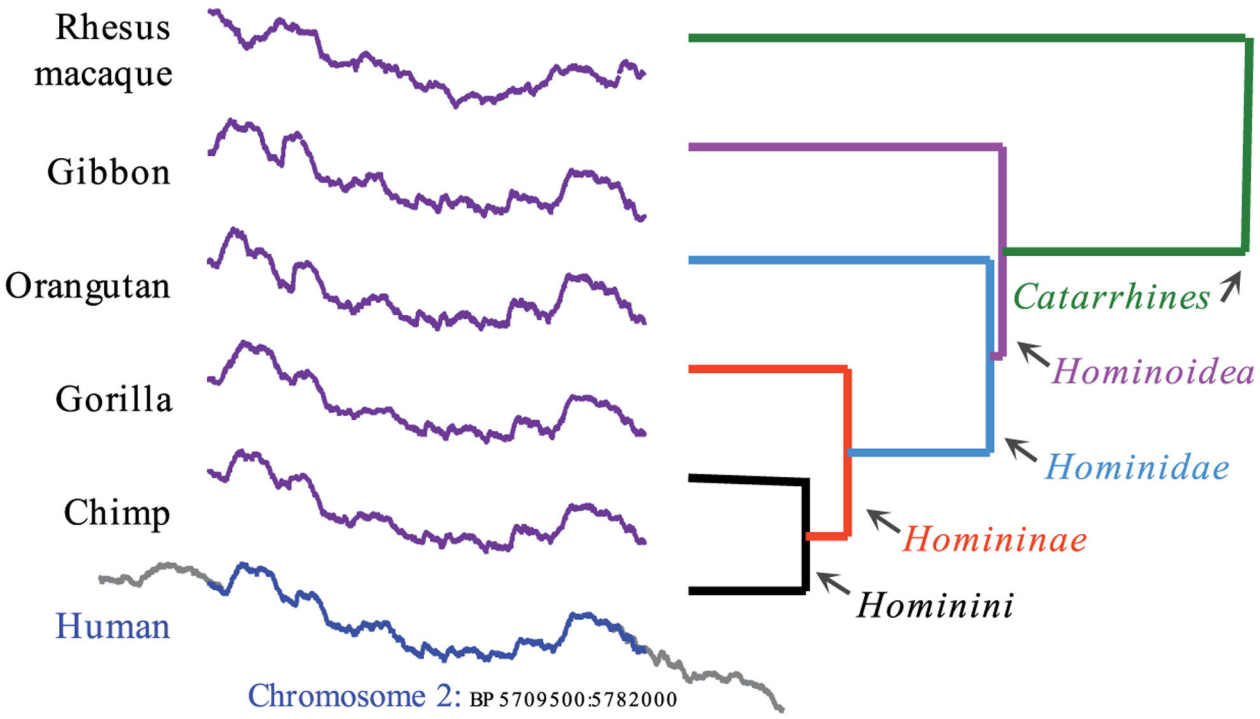


Fig. 14.

A subsequence of DNA from Human chromosome 2, of length 72,500 and beginning at 5,709,500, is clustered using single linkage with its Euclidean distance nearest neighbors from five other primates.

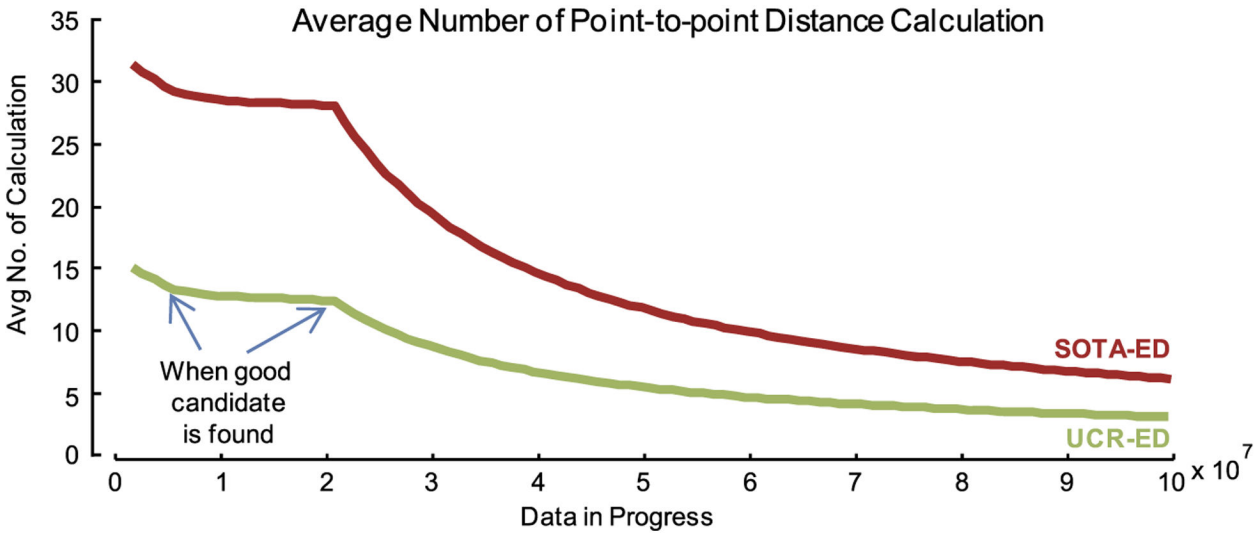
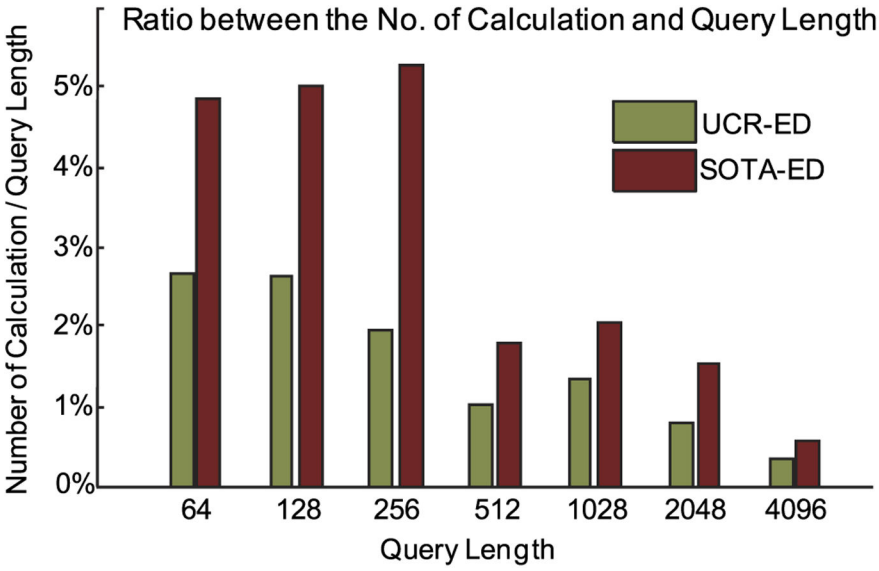


Fig. 15.

Typical numbers of point-to-point distance calculations from SOTA-ED and UCR-ED. In this query of length 128, our UCR suite calculates only 3.34 times per datapoint on average. Note that this number is reduced when the data is longer.



No. of Calculation per one Data Point

Q Len	UCR-ED	SOTA-ED
64	1.694	3.088
128	3.349	6.456
256	4.959	13.491
512	5.219	9.118
1024	13.47	20.925
2048	16.306	31.37
4096	14.102	22.741

Fig. 16.

The number of point-to-point calculations in a nearest neighbor search with different query lengths. (right) The number of calculations per single datapoint. (left) The number of calculations shown as a fraction of the query length. Unintuitively, this ratio is typically reduced when the query length is longer. At query length 4,096, the UCR suite computes only 14.1 calculations per datapoint, or just 0.3% of the query length.

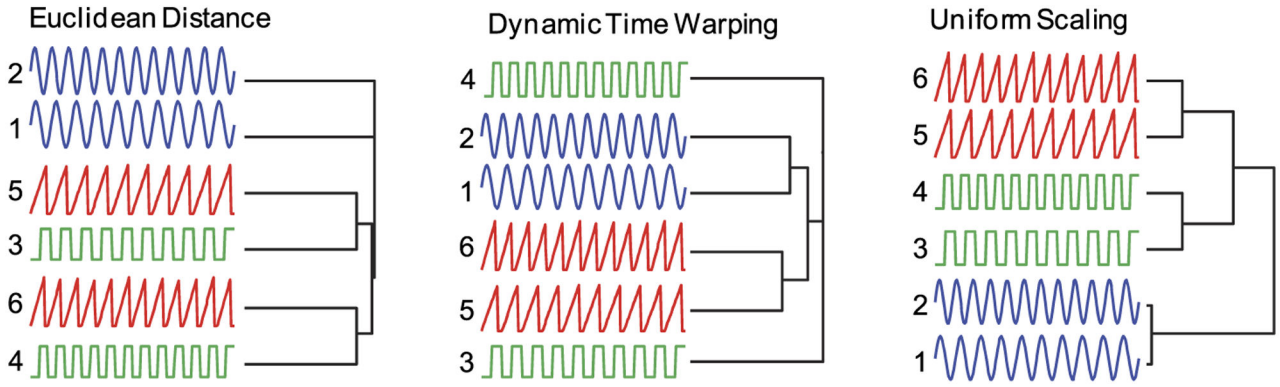


Fig. 17.

A clustering of synthetic time series using (left) Euclidean distance, (middle) DTW with a band size of 20%, (right) uniform scaling with a maximum scaling factor of 20%. Only uniform scaling finds the correct clustering.

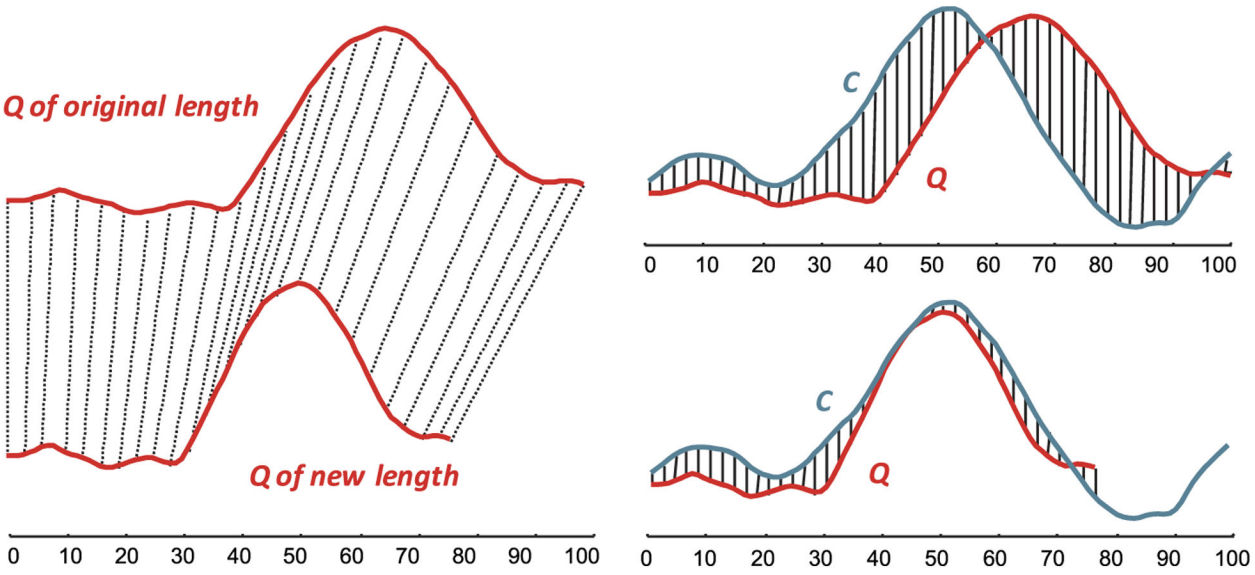


Fig. 18.

(left) The subsequence Q is scaled down from the original scale to a smaller scale. (top-right) The distance, called *Euclidean distance*, between subsequence Q and C in the original scale. (bottom-right) The distance, called *uniform scaling distance*, can be much smaller than the normal Euclidean distance. In this example, Q is scaled to length 77. Note that both Q and C are the same as the example in Section 3 but are in the reverse order.

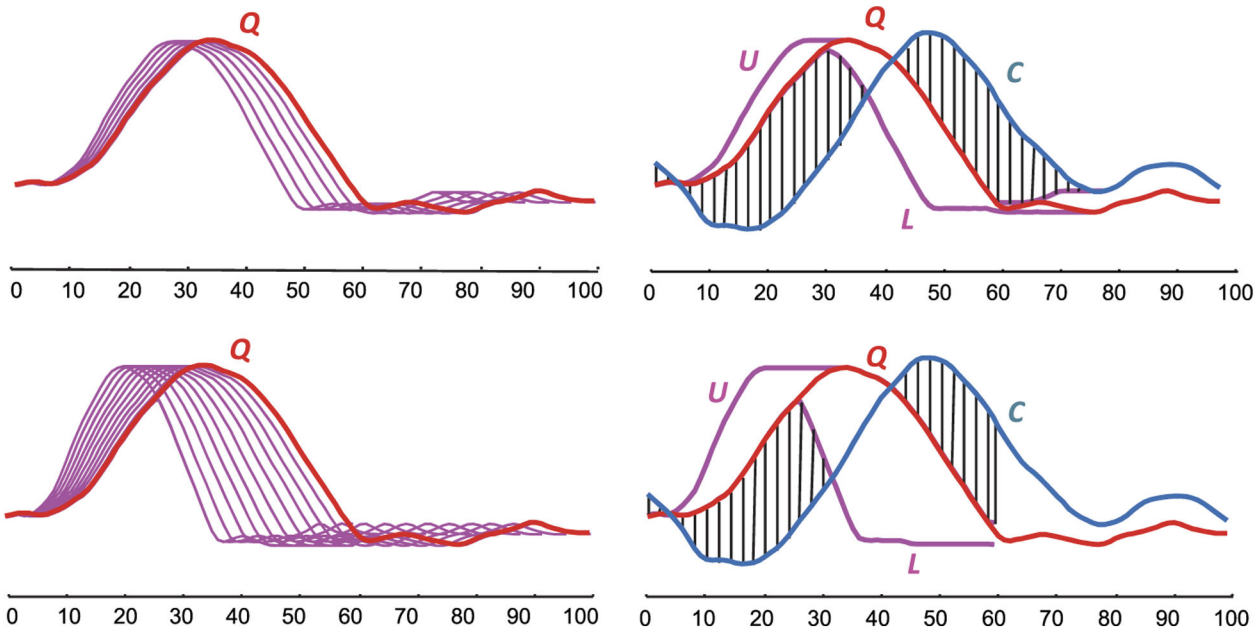


Fig. 19.

(top-left) All scaling subsequences created from the query Q scaled to different lengths from 80 to 100. (top-right) The upper envelope U and lower envelope L created by the maximum and minimum values of all scaling subsequences in (left). The lower bound distance is computed from the candidate C to the closer envelope, as shown in vertical lines. (bottom) The query is scaled from 60 to 100, or has a scaling factor of 40%. Note that although the scaling factor may be large, in most cases, the lower bound is still large and can prune candidates efficiently.

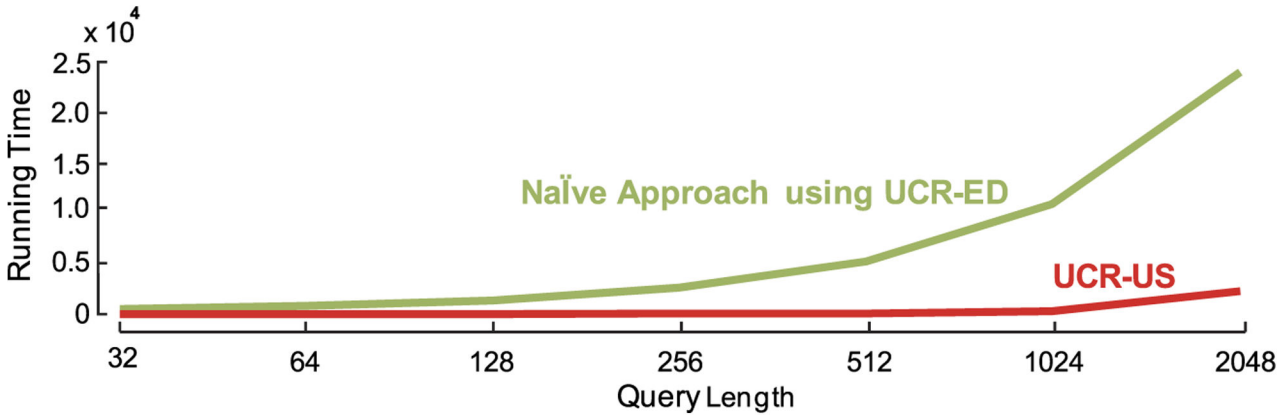


Fig. 20.

The running time of uniform scaling nearest neighbor when the query is varied from length 32 to 2048.

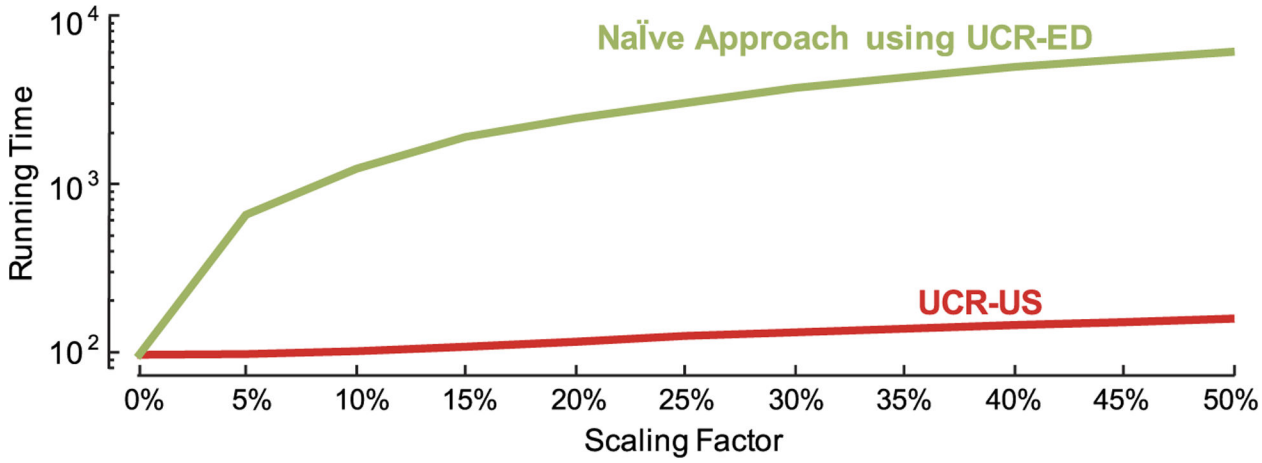


Fig. 21.

The running time of uniform scaling nearest neighbor when the scaling factor is varied from 0% to 50%. Note that, when the scaling factor is 0%, it is simply ED nearest neighbor and when the scaling/shrinking factor is 50% it means that the answer match can be twice as short as the query length.

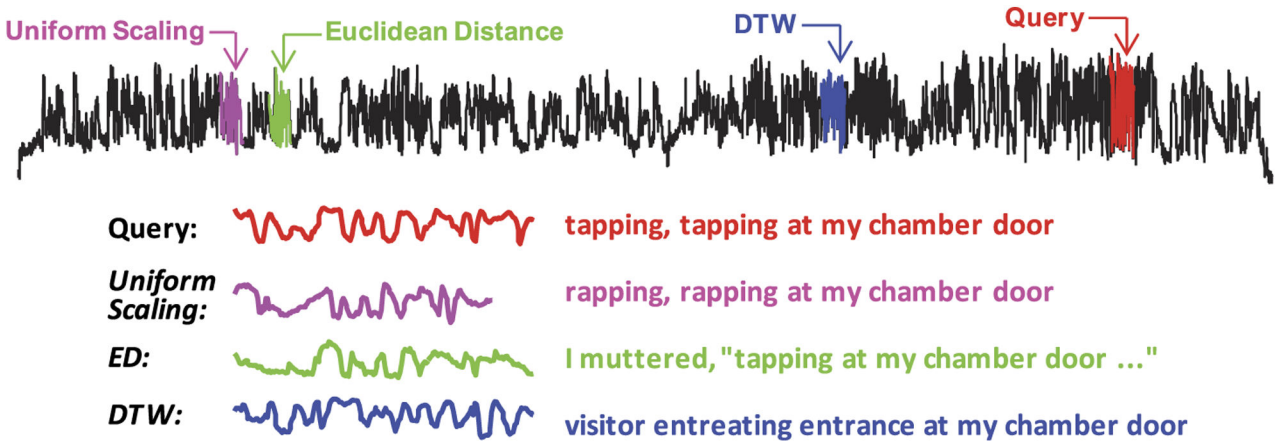


Fig. 22.

(top) Time series created by MFCC conversion of the audio of the poem *The Raven*, by Edgar Allan Poe. The query, the nearest neighbor using DTW, Euclidean distance, and uniform scaling are presented in red, blue, green, and magenta, respectively. (bottom) The same subsequences in the top and their corresponding text in the poem. Note that the scaling factor for uniform scaling and warping windows for DTW are set to 10%.

Table I.

Subsequence Search with Online Z-Normalization

Algorithm	Similarity Search
Procedure	[*nn*] = *SimilaritySearch(T,Q)*
1	*best-so-far* ← ∞, *count* ← 0
2	*Q* ← *Z-normalize(Q)*
3	**while** !*next(T)*
4	*i* ← *mod(count,m)*
5	*X*[*i*] ← *next(T)*
6	*ex* ← *ex* + *X[i]*, *ex*2 ← *ex*2 + *X*[*i*]^2
7	**if** *count* ≥ *m* − 1
8	μ ← *ex*/*m*, σ ← *sqrt*(*ex2*/*m* − μ^2)
9	*j* ← 0, *dist* ← 0
10	**while** *j* < *m* **and** *dist* < *best-so-far*
11	*dist* ← *dist* + (*Q*[*j*] − (*X*[*mod*(*i* + 1 + *j*, *m*)] − μ)/σ)^2
12	*j* ← *j* + 1
13	**if** *dist* < *best-so-far*
14	*best-so-far* ← *dist*, *nn* ← *count*
15	*ex* ← *ex* − *X*[*mod*(*i* + 1,*m*)]
16	*ex*2 ← *ex*2 − *X*[*mod*(*i* + 1,*m*)]^2
17	*count* ← *count* + 1

Table II.

Time Taken to Search a Random Walk Dataset With $|Q|= 128$

	Million (*Seconds*)	Billion (*Minutes*)	Trillion (*Hours*)
UCR-ED	0.034	0.22	3.16
SOTA-ED	0.243	2.40	39.80
UCR-DTW	0.159	1.83	34.09
SOTA-DTW	2.447	38.14	472.80

Table III.

Time to Search 303,523,721,928 EEG Datapoints, $|Q|$= 7000

Note that only ED is considered here because DTW may produce false positives caused by eye blinks		**UCR-ED**	**SOTA-ED**
	EEG	3.4 hours	494.3 hours

Table IV.

An Algorithm to Convert DNA to Time Series

$T_1 = 0$,

for i = 1 **to** |DNAstring|

if $DNAstring_i$ = **A**, **then** $T_{i+1} = T_i + 2$

if $DNAstring_i$ = **G**, **then** $T_{i+1} = T_i + 1$

if $DNAstring_i$ = **C**, **then** $T_{i+1} = T_i - 1$

if $DNAstring_i$ = **T**, **then** $T_{i+1} = T_i - 2$

Table V.

Time Taken to Search One Year of ECG Data with $|Q|$= 421

	UCR-ED	SOTA-ED	UCR-DTW	SOTA-DTW
ECG	4.1 minutes	66.6 minutes	18.0 minutes	49.2 hours

Table VI.

UCR Suite: Uniform Scaling Search

```
Algorithm    UCR Suite: UniformScaling
Procedure    [nn] = UniformScalingSearch(T,Q,scaling_factor)
1    best-so-far ← ∞
2    max_len ← length(Q)
3    while !next(T)
4      C ← current candidate from T of length n
5      max_len ← max_len*(1-scaling_factor)
6      [U, L] ← CreateEnvelope(Q, min_len, max_len)
7      lb ← LB_Keogh(U,L,C)
8      if lb < best-so-far
9        for len ← min_len to max_len
10         Qs ← ScalingSubsequence(Q,len)
11         dist ← UCR_ED(Qs,C)
12         if dist < best-so-far
13           best-so-far ← dist
14           nn.sequence ← C
15           nn.best_len ← len
16         end if
17       end for
18     end if
19   end while
```