Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping

2012/12/6

Kuan-Hua Lin (Khlin@netdb.cs.nthu.edu.tw)
NetDB, NTHU, Taiwan

KDD 2012, Best Paper Award, 4 cited

Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, Eamonn Keogh
(UC Riverside, Brigham and Women's Hospital, University of São Paulo)

Outline

- Introduction
- Explicit Statement of our Assumptions
- Background and Notations
- Known Optimizations
- Novel Optimizations: The UCR Suite
- Experimental Results
- Conclusion

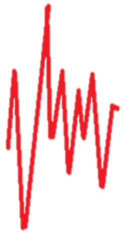
Outline

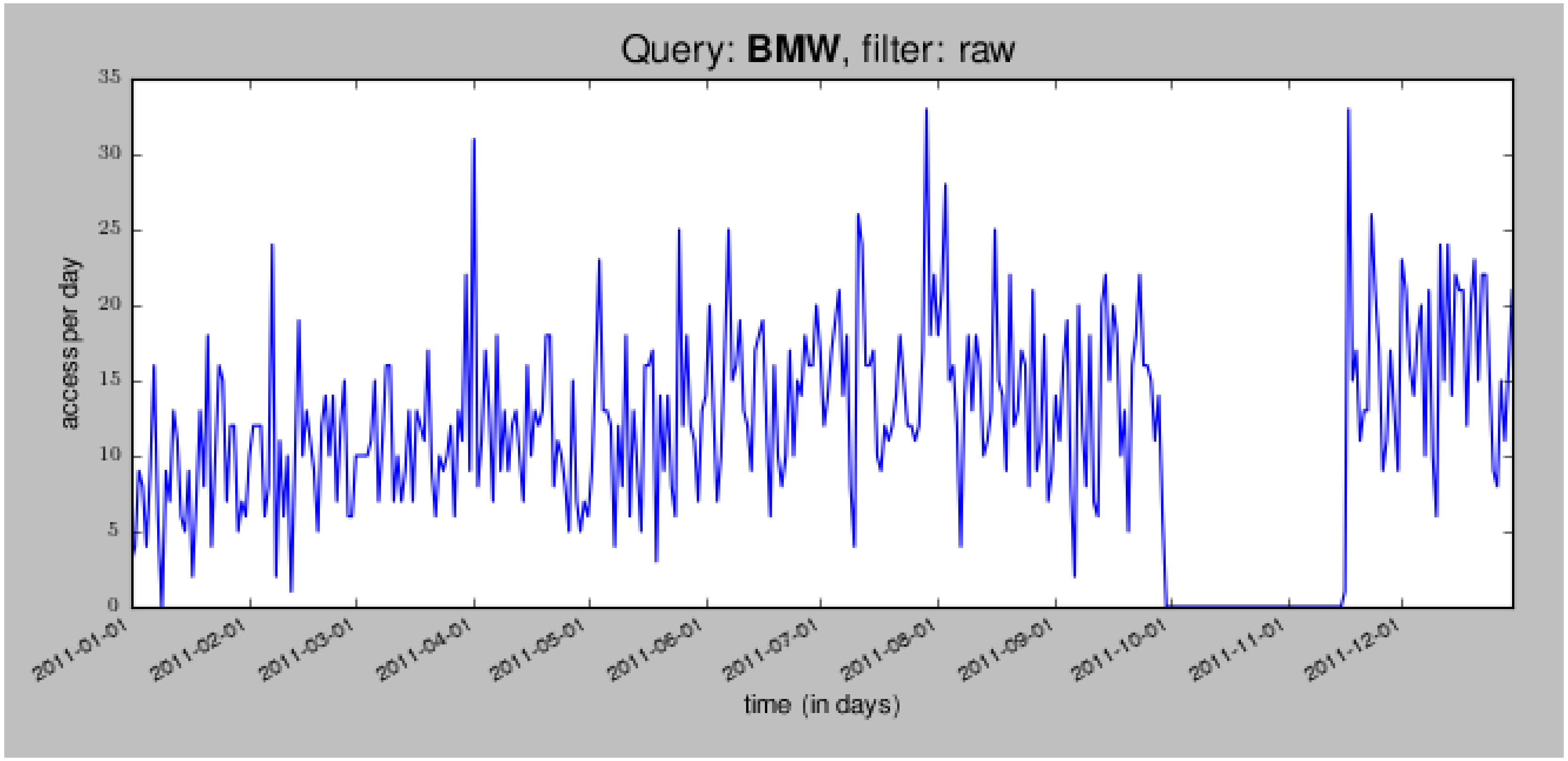
- Introduction
- Explicit Statement of our Assumptions
- Background and Notations
- Known Optimizations
- Novel Optimizations: The UCR Suite
- Experimental Results
- Conclusion

Introduction

- Time series data mining has attracted significant attention
  - Time series data is pervasive
  - Ex: medicine, finance, science and entertainment
- Most time series data mining algorithms use similarity search as a core subroutine, and the similarity search is the time bottleneck
  - It is difficult to scale search to large datasets , so most academic work on time series data mining consider **a few millions** of time series objects
- However, much of industry and science sits on **billions** of time series objects waiting to be explored

Example

- Find the subsequence most similar to 



Introduction

- There is increasing evidence that the classic Dynamic Time Warping (DTW) measure is the best measure in most domains
  - However, it's too computationally expensive
  - Ex: "*Ideally, dynamic time warping would be used to achieve this, but due to time constraints…*" [5]
- Propose four ideas to speed up DTW
  - Much faster than any current *approximate* search or exact *indexed* search
- **Mining truly massive (trillions) time series for the first time**

A Brief Disscussion of Trillion

- The word “trillion” has never appeared in a data mining/database paper
- Trillion: 1 million million, 10^{12}, or 1,000,000,000,000
  - If we have a time series T of length one trillion, with eight bytes of each value, it will require 7.2 terabytes to store
- **It’s more than all time series data considered in all papers in all data mining conferences *combined***
  - Up to 2011 there have been 1,709 KDD/SIGKDD papers, If *every* paper was on time series, and *each* had looked at 500 million objects, this would still not add up to a trillion
  - However, the largest time series data considered in a SIGKDD paper was a “mere” 100 million objects

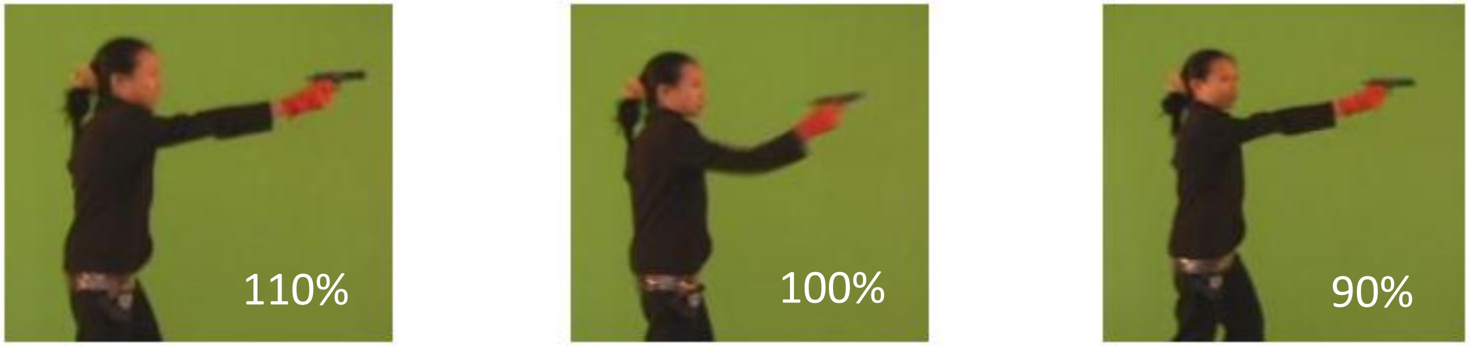
Outline

- Introduction
- Explicit Statement of our Assumptions
- Background and Notations
- Known Optimizations
- Novel Optimizations: The UCR Suite
- Experimental Results
- Conclusion

Assumption 1

- **Time Series Subsequences must be Normalized**
  - In order to make meaningful comparisons between two time series, both must be normalized

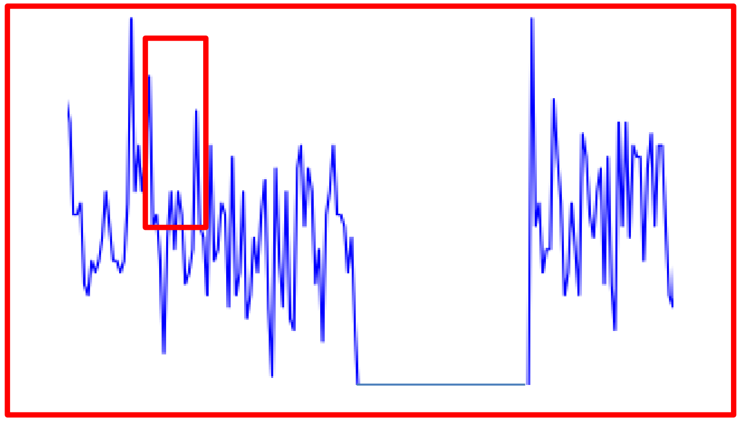
Gun/NoGun data



  - The problem has a 50/150 train/test split and a DTW one-nearest-neighbor classifier achieves an error rate of 0.087
  - For the unnormalized data, error rates are 0.326 and 0.193, respectively, significantly worse than the normalized case

Assumption 1

- The apparent *scale* can be changed by
  - The camera zooming
  - The actor standing a little closer to the camera
  - An actor of a different height.
  - ….
- We must normalize *each* subsequence before making a comparison, it is not sufficient to normalize the entire dataset.



Assumption 2

- **Dynamic Time Warping is the Best Measure**
  – 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 [6]
  – Thus, DTW is *the* measure to optimize

Assumption 3

- **Arbitrary Query Lengths cannot be Indexed**
  – If we know the length of queries ahead of time we can improve the search by indexing the data
  – But we are interested in a trillion data objects …
  – Moreover, if we are able to build indexes that are not too different in size
    - Ex: {2, 4, 8, 16, 32, 64, …}
    - Suppose we have a query Q of length 65
    - We search the index 64 for $Q[1:64]$ and find that the best match
    - But is it also the best match of Q?

Assumption 4

- **There Exists Data Mining Problems that we are Willing to Wait Some Hours to Answers**
  - This point is almost self-evident
  - groups would be willing to spend hours of CPU time to glean knowledge from their data.
    - If a team of entomologists has spent three years gathering 0.2 trillion datapoints
    - If astronomers have spent billions dollars to launch a satellite to collect one trillion datapoints of star-light curve data per day
    - If a hospital charges $34,000 for a daylong EEG session to collect 0.3 trillion datapoints

Outline

- Introduction
- Explicit Statement of our Assumptions
- Background and Notations
- Known Optimizations
- Novel Optimizations: The UCR Suite
- Experimental Results
- Conclusion

Time Series and Subsequence

- **Definition 1:** A *Time Series T* is an ordered list: $T = t_1, t_2, \dots, t_m$
- While the source data is a long time series, we 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 . $\mathrm{n} = |Q|$.

Euclidean Distance

- **Definition 3:** The Euclidean distance (ED) between Q and C, where $|Q| = |C|$, is defined as: $ED(Q, C) = \sqrt{\Sigma_{i=1}^{n}(q_i - c_i)^2}$

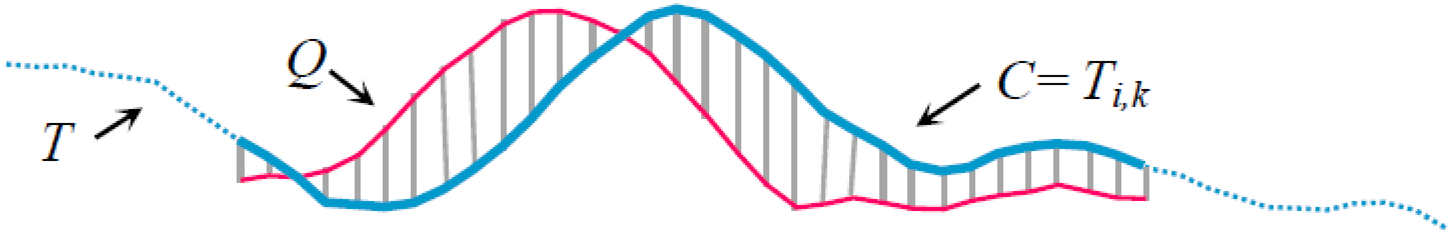


Figure 2: A long time series T can have a subsequence $T_{i,k}$ extracted and compared to a query Q under the Euclidean distance, which is simply the square root of the sum of the squared hatch line lengths

Dynamic Time Warping (1/3)

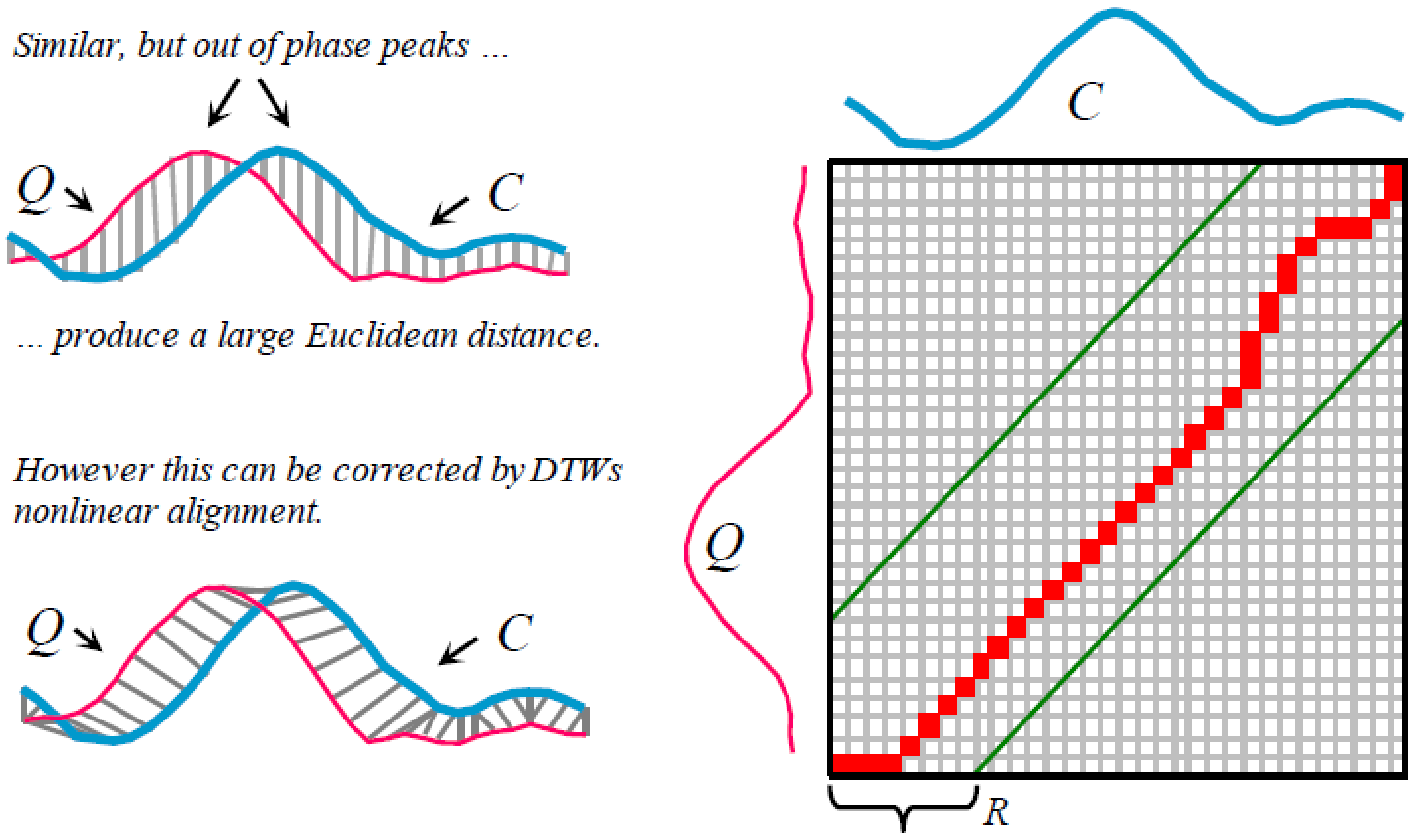
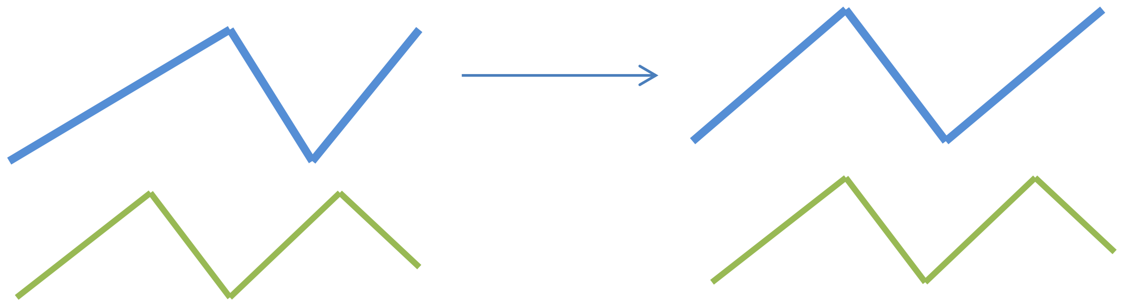

Similar, but out of phase peaks …
Q
C
… produce a large Euclidean distance.
However this can be corrected by DTWs nonlinear alignment.
Q
C
C
Q
R

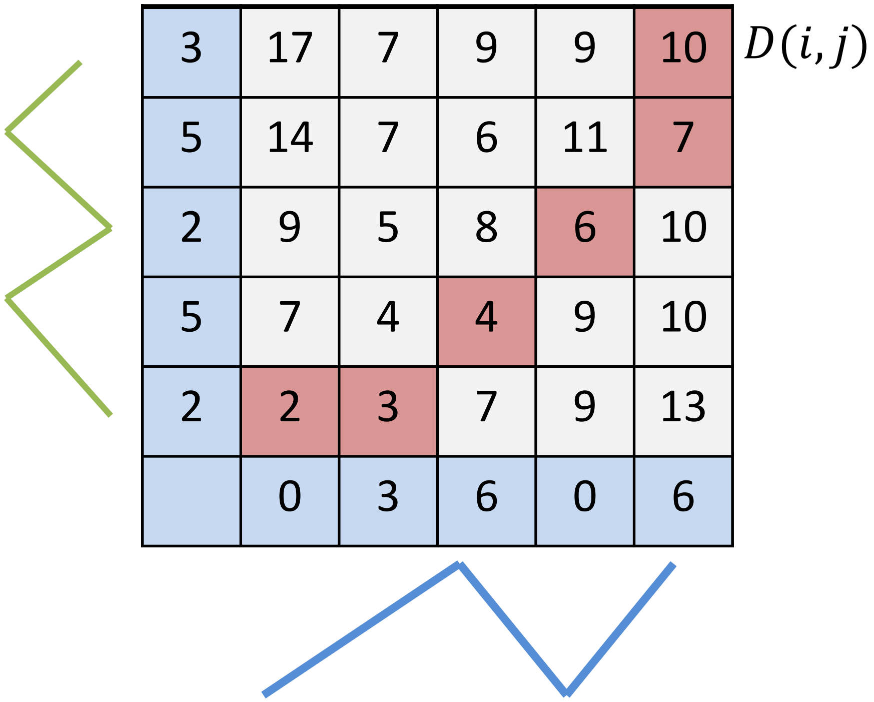
Figure 3: ***left*) Two time series which are similar but out of phase. *right*) To align the sequences we construct a warping matrix, and search for the optimal warping path (red/solid squares). Note that Sakoe-Chiba Band with width *R* is used to constrain the warping path**

Dynamic Time Warping (2/3)

- **Dynamic time warping** (DTW) is an algorithm for measuring similarity between two sequences which may vary in time or speed.
  - For instance, similarities in walking patterns would be detected, even if in one video the person was walking slowly and if in another he or she were walking more quickly



Dynamic Time Warping (3/3)



3	17	7	9	9	10
5	14	7	6	11	7
2	9	5	8	6	10
5	7	4	4	9	10
2	2	3	7	9	13
	0	3	6	0	6

$$D(i,j) = d(i,j) + \min\begin{Bmatrix} D(i-1,j-1) \\ D(i-1,j) \\ D(i,j-1) \end{Bmatrix}$$

Where $d(i,j)$ is the Euclidean distance

- Suppose we have two sequences to be compared C, Q
- Construct a $|C| \times |Q|$ distance table
- A warping path W is then calculated
  - Start at $(1,1)$ and end at $(|C|, |Q|)$

Outline

- Introduction
- Explicit Statement of our Assumptions
- Background and Notations
- Known Optimizations
- Novel Optimizations: The UCR Suite
- Experimental Results
- Conclusion

Using the Squared Distance

- Both DTW and ED have a square root calculation.
- If we omit this step, it does not change the relative rankings of nearest neighbors
  - Note that this is only an internal change in the code. After finding the qualifying objects, we can take the square root of the distances for the qualifying objects and present to the user.
- Where there is no ambiguity below, we will still use ‘DTW’ and ‘ED’; however, the reader may assume we mean the squared versions of them

Lower Bounding

- A classic trick to speed up is to use a lower bound to prune off unpromising candidates

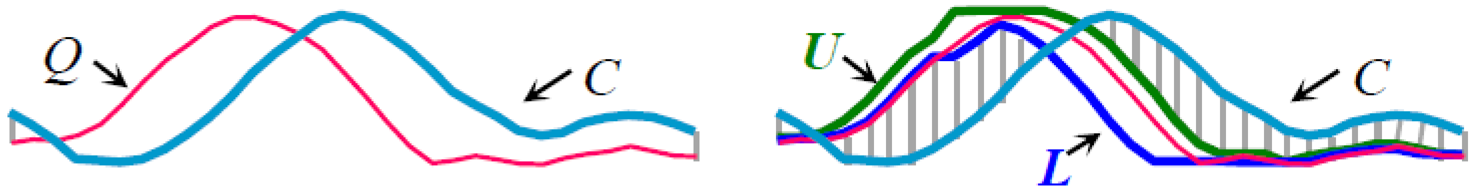


Figure 4: *left*) The $LB\_{}_{Kim}FL$ lower bound is $O(1)$ and uses the distances between the First (Last) pair of points from C and Q as a lower bound. It is a simplification of the original $LB\_{}_{Kim}$ [21]. *right*) The $LB\_{}_{Keogh}$ lower bound is $O(n)$ and uses the Euclidean distance between the candidate sequence C and the closer of $\{U,L\}$ as a lower bound

Early Abandoning of ED and LB_{Keogh}

- During the computation of the ED or LB_{Keogh} lower bound, if the current sum of the squared differences exceeds the *best-so-far*, then we can stop the calculation

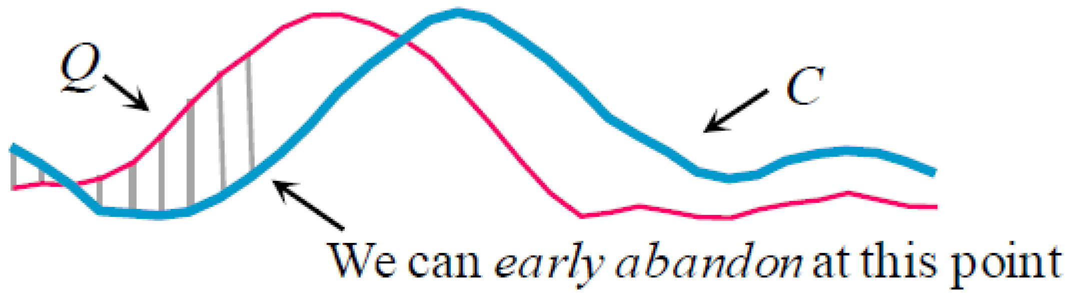
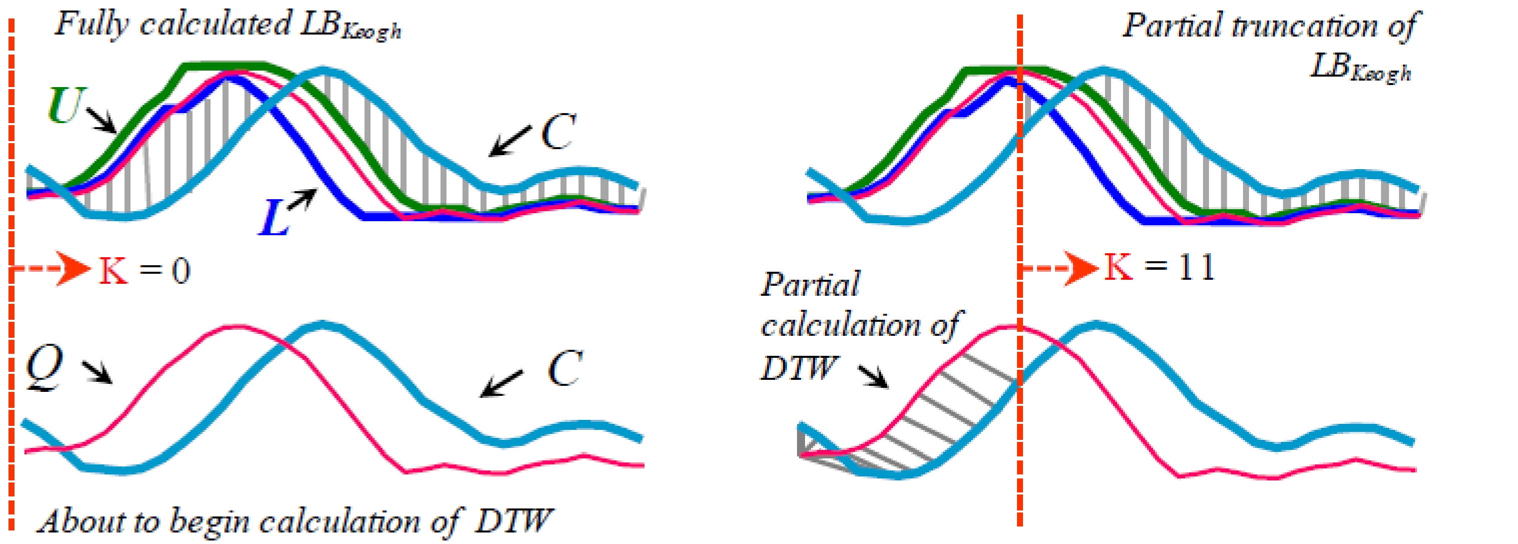


Figure 5: An illustration of ED *early abandoning*. We have a *best-so-far* value of *b*. After incrementally summing the first nine (of thirty-two) individual contributions to the ED we have exceeded *b*, thus it is pointless to continue the calculation [20]

Early Abandoning of DTW

- If we must calculate the full DTW
- Incrementally compute the DTW from left to right, (from 1 to K), sum the *partial* DTW accumulation with the LB_{Keogh} accumulation from K+1 to *n*.
- Stop and abandon if exceed the *best-so-far*



Fully calculated LB_{Keogh}
U
C
L
K = 0
Q
C
About to begin calculation of DTW
Partial truncation of LB_{Keogh}
K = 11
Partial calculation of DTW

Exploiting Multicores

- We can get essentially linear speedup using multicores
- The *software* improvements presented in the experiment **completely dwarf the improvements gained by multicores**
  - 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*" [34].
  - Using ideas in this paper, it costs under 1s on a single core.
  - Nevertheless, as it is simple to port to the now ubiquitous multicores, we consider them below

Outline

- Introduction
- Explicit Statement of our Assumptions
- Background and Notations
- Known Optimizations
- Novel Optimizations: The UCR Suite
- Experimental Results
- Conclusion

Novel Optimizations: The UCR Suite

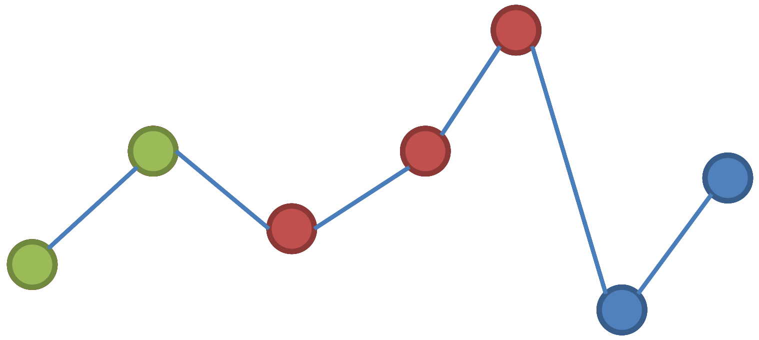
- Propose four optimizations of search under ED and/or DTW.

Early Abandoning Z-Normalization (1/2)

- No one has ever considered optimizing the *normalization* step
  - Surprising since it takes slightly longer than computing the Euclidean distance itself.
- Z-normalization: $z_i = \frac{x_i - \mu}{\sigma}, i \in \mathbb{N}$
  - Recall that the mean and standard deviation of a sample can be computed from the sums of the values and their squares $\mu = \frac{1}{m}\Sigma x_i, \sigma^2 = \frac{1}{m}\Sigma x_i^2 - \mu^2$
  - Incrementally compute the Z-normalization, and stop if early abandon

Early Abandoning Z-Normalization (2/2)

- In similarity search, every subsequence needs to be normalized before it is compared to the query
- They can be obtained by keeping two running sums of the long time series which have a lag of exactly m values.
  - $\mu = \frac{1}{m}\left(\Sigma_{i=1}^{k} x_i - \Sigma_{i=1}^{k-m} x_i\right)$,
  - $\sigma^2 = \frac{1}{m}\left(\Sigma_{i=1}^{k} x_i^2 - \Sigma_{i=1}^{k-m} x_i^2\right) - \mu^2$



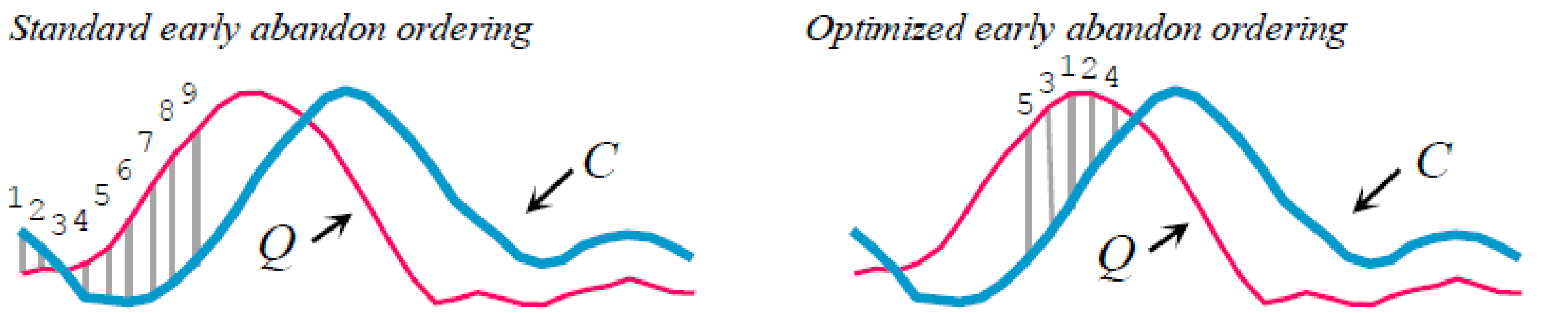
Query length = 3

$\Sigma(1 \approx 5) = \Sigma(1 \sim 2)$

$\Sigma(1 \approx 4) = \Sigma(1)$

Reordering Early Abandoning (1/2)

- In early abandoning method, is incrementally compute the distance/normalization from left to right the best ordering?
  - Different orderings produce different speedups

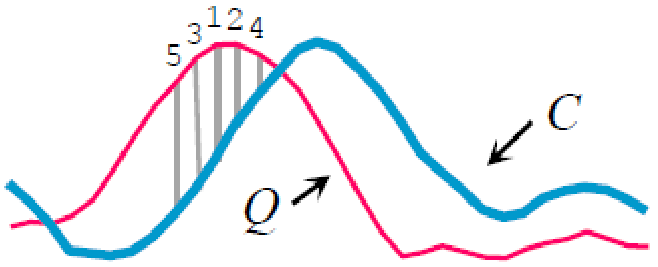


Standard early abandon ordering
1 2 3 4 5 6 7 8 9
Q
C
Optimized early abandon ordering
5 3 1 2 4
Q
C

Figure 7: ***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**

Reordering Early Abandoning (2/2)

- Is there a *universal* optimal ordering?
  - $n!$ possible orderings to consider
- Conjecture: the universal optimal ordering is to sort based on the absolute values of the Z-normalized Q.
  - Intuition: 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 farthest from the mean, zero, will *on average* have the largest contributions to the distance measure
  - Empirically, the conjecture is true



Reversing the Query/Data Role in LB_{Keogh}

- The LB_{Keogh} lower bound builds the envelope around the *query*
  - Only needs to be done once, and thus saves the time and space
- Built the envelope around *candidate,* too

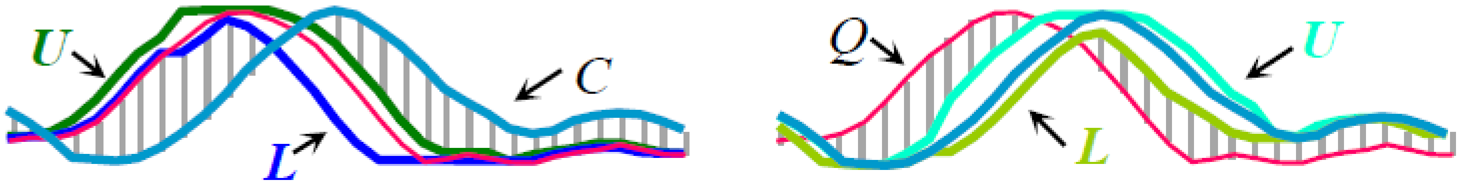


Figure 8: *left*) Normally the LB_{Keogh} envelope is built around the query (see also Figure 4.*right*), and the distance between C and the closer of $\{U,L\}$ acts as a lower bound. *right*) However, we can reverse the roles such that the envelope is built around C and the distance between Q and the closer of $\{U,L\}$ is the lower bound

Cascading Lower Bounds (1/2)

- One way to speed up is using lower bounds to prune off unpromising candidates
- it is hard to state which is the best bound to use
  - there is a tradeoff between the tightness of the lower bound and how fast it is to compute
- Implement all published lower bounds, as followed

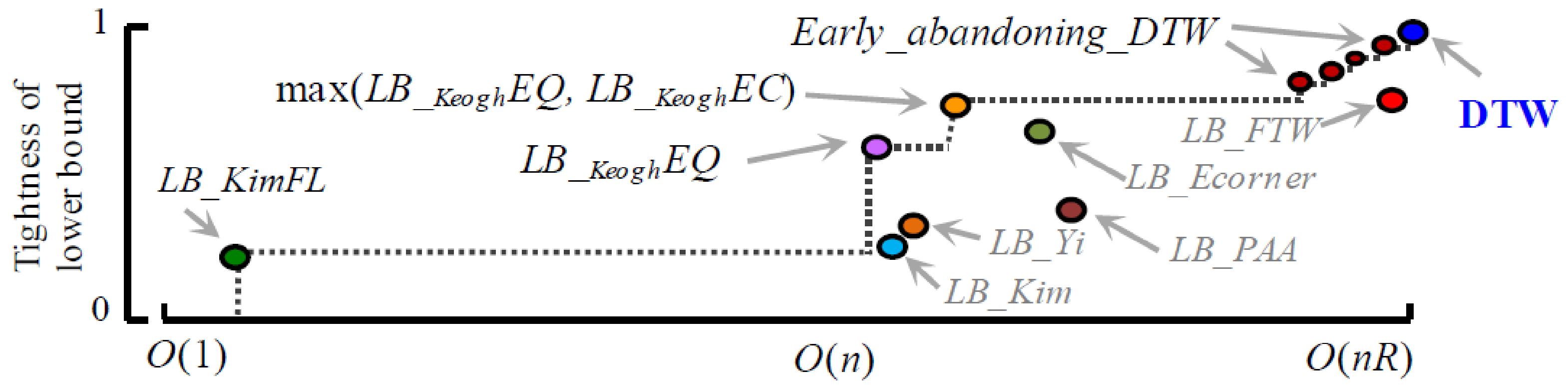


Figure 9: The mean tightness of selected lower bounds from the literature plotted against the time taken to compute them

Cascading Lower Bounds (2/2)

- Which of the lower bounds should we use?
- We should use *all* of them in a cascade.
  - First use the $O(1)$ LB_{KimFL}, which while a very weak lower bound prunes many objects.
  - If a candidate is not pruned, compute the $LB_{KeoghEQ}$
  - If it does not exceed the *best-so-far*, we reverse the query/data role and compute $LB_{KeoghEC}$
  - If this bound does not allow us to prune, then start the early abandoning calculation of DTW
- More than 99.9999% of DTW calculations for a large-scale search can be pruned using this technique

Outline

- Introduction
- Explicit Statement of our Assumptions
- Background and Notations
- Known Optimizations
- Novel Optimizations: The UCR Suite
- Experimental Results
- Conclusion

Source Code

- Our experiments are reproducible.
- All data and code are available in perpetuity
  - http://www.cs.ucr.edu/~eamonn/UCRsuite.html

Compared Methods

- Consider the following methods
  - Naive
    - Each subsequence is Z-normalized from scratch. The full ED or the DTW is used at each step.
    - About 2/3 of the papers in the literature do 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
    - About 1/3 of the papers in the literature do this
  - UCR Suite
    - Use all of the speedup techniques in this paper

Implementation

- It is critical to note that our implementations of Naive, SOTA are incredibly efficient and tightly optimized, and they are not “crippled” in any way.
  - Ex: we could implement SOTA recursively rather than iteratively, and that would make SOTA at least an order of magnitude slower.
- DTW uses R = 5% unless otherwise noted.
- For experiments where Naive or SOTA takes more than 24 hours to finish, we present the interpolated values, shown in color gray

Baseline Tests on Random Walk (1/3)

- We begin with experiments on random walk data
  - Random walks model financial data very well and are often used to test similarity search schemes
- 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.

Baseline Tests on Random Walk (2/3)

- In Table 2 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

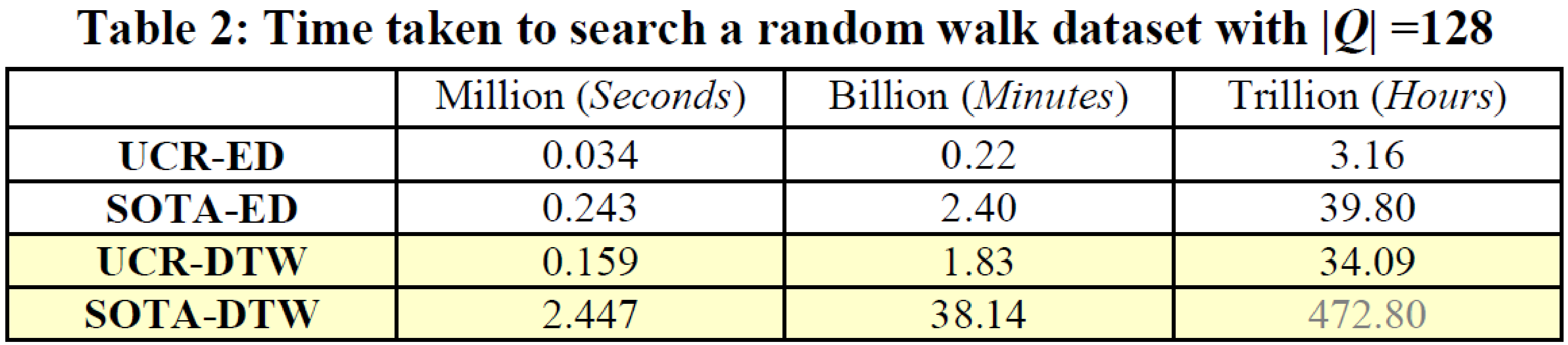


Table 2: Time taken to search a random walk dataset with $|Q| = 128$

	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

Baseline Tests on Random Walk (3/3)

- What happens if we consider longer queries?

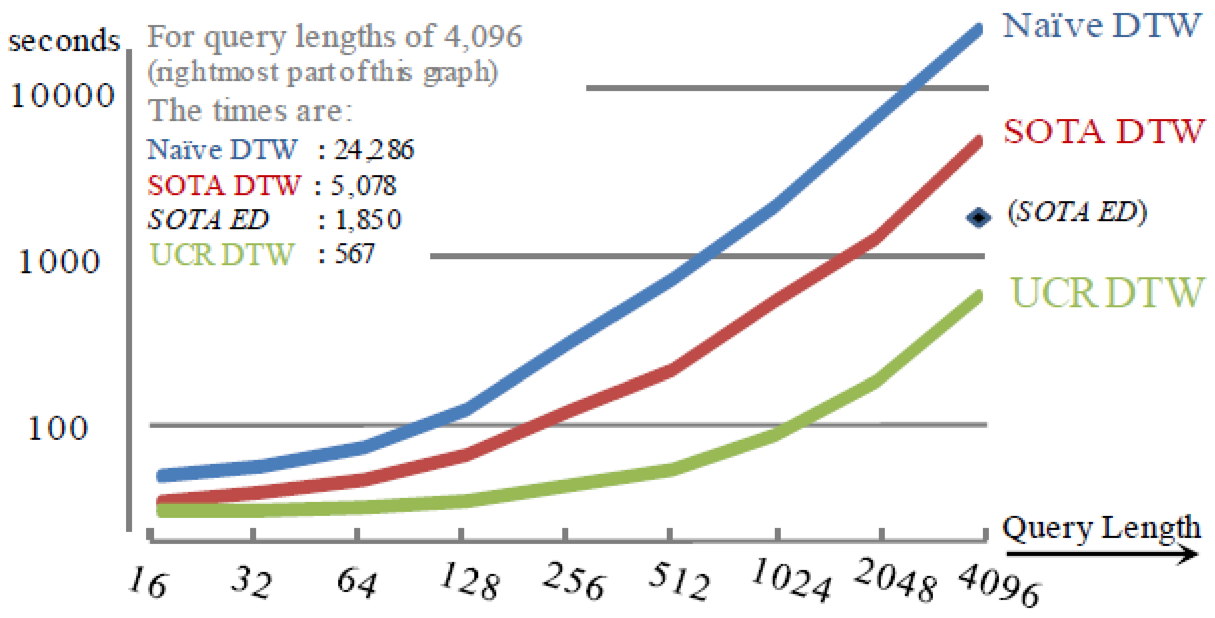
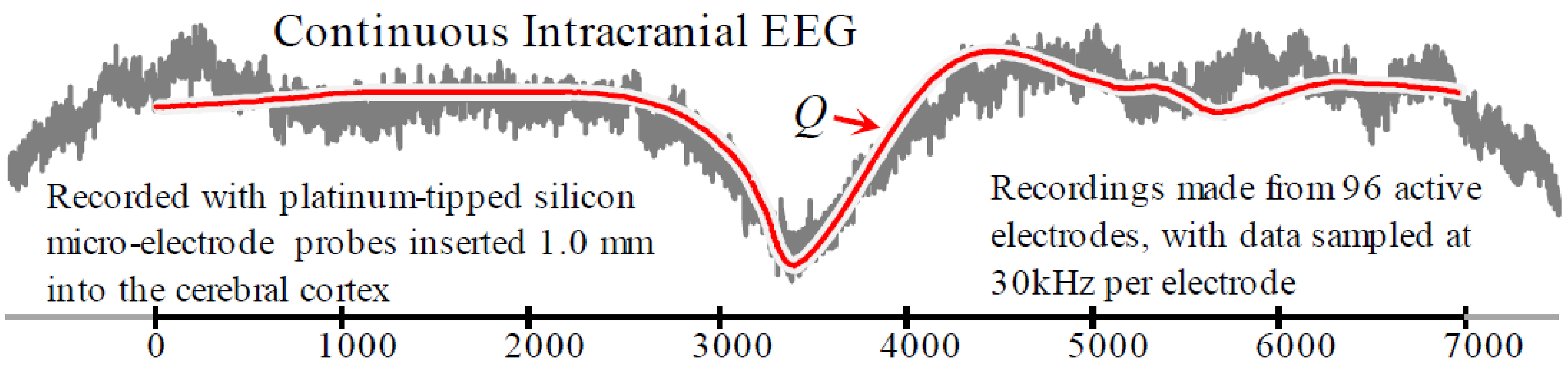


Figure 10: The time taken to search random walks of length 20 million with increasingly long queries, for three variants of DTW. In addition, we include just length 4,096 with SOTA-ED for reference

For queries of length 4,096, UCR-DTW is even faster than SOTA ED.
(To reduce visual clutter we have only placed one ED value)

Supporting Long Queries: EEG

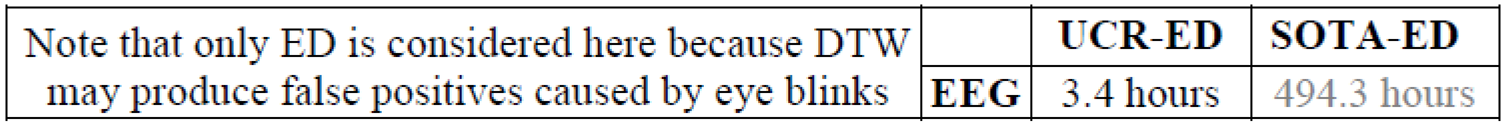
- Even though 4,096 is longer than any published query lengths in the literature, there is a need for even *longer* queries.
- The first user of the UCR suite was Dr. Sydney Cash, who wants to search massive archives of EEG data for examples of epileptic spikes



Supporting Long Queries: EEG

- From a single patient S.C. gathered 0.3 trillion datapoints and searched for a prototypical epileptic spike Q he created by averaging spikes from other patients.
  - The query length was 7,000 points (0.23 seconds).

Table 3: Time to search 303,523,721,928 EEG datapoints, $|Q| = 7000$



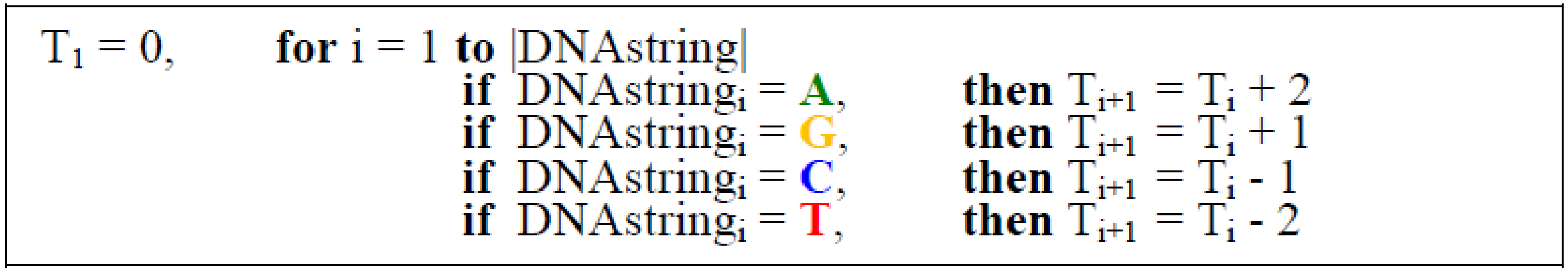
Note that only ED is considered here because DTW may produce false positives caused by eye blinks		**UCR-ED**	**SOTA-ED**
	EEG	3.4 hours	494.3 hours

  - This data took multiple sessions over seven days to collect, at a cost of approximately $34,000 [43], so the few hours of CPU time we required to search the data are dwarfed in comparison.

Supporting Very Long Queries: DNA (1/3)

- 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 1,024 data points 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

Table 4: 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 $T_{i+1} = T_i - 2$
```

Supporting Very Long Queries: DNA (2/3)

- Experiment with a section of Human chromosome 2 (H2) to. 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

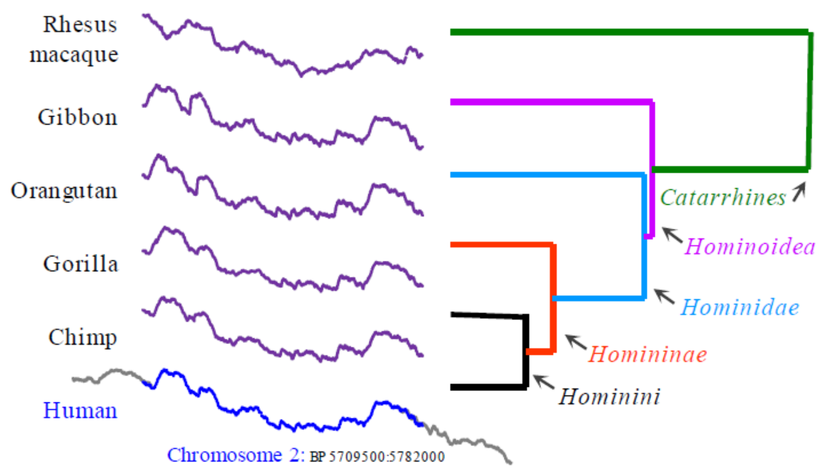


Figure 12: A subsequence of DNA from Human chromosome 2, of length 72,500 beginning at 5,709,500 is clustered using single linkage with its Euclidean distance nearest neighbors from five other primates

- The clustering *is* the correct grouping
- Query length = 72,500, and the genome chimp is 2,900,629,179 base pairs in length.
- The single-core nearest neighbor search using Naïve took 38.7 days, **34.6 days** using SOTA, but only **14.6 hours** using the UCR suite.

Supporting Very Long Queries: DNA (3/3)

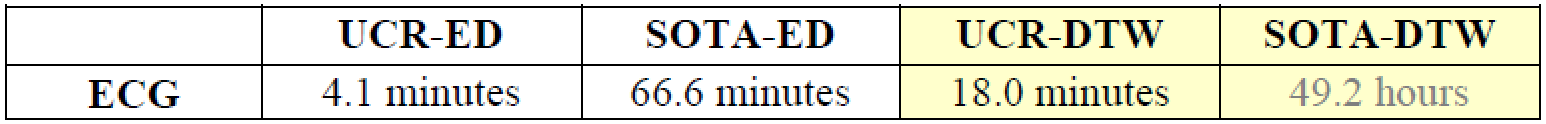
Realtime Medical and Gesture Data (1/2)

- 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 [22].
- There are dozens of research efforts in this domain that explicitly state that while monitoring under DTW is *desirable*, it is impossible
  - "*still too slow for gesture recognition systems*" [9]
  - "*DTW is unusable for real-time recognition purposes*" [12]
  - "*Processing of one hour of speech using DTW takes a few hours*." [34]

Realtime Medical and Gesture Data (2/2)

- Created a diverse dataset of one year of electrocardiograms (ECGs) sampled at 256Hz with 8,518,554,188 datapoints by concatenating the ECGs of more than 200 people.
- USC cardiologist Dr. Helga Van Herle helped to produce an idealized Premature Ventricular Contraction (PVC) query.
- Run the experiment on a multi-core desktop
- The results are 29,219 times faster than real-time (256Hz)
  – (8518554188/ (18*60)) Hz / (256) Hz
- So, real-time DTW is tenable even on low-power devices.

Table 5: 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

Speeding up Existing Mining Algorithms (1/2)

- 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 already try to prune as many distance calculations as possible
  - Nevertheless, even though the speedups are relatively small (1.5X to 16X), they are “free”, requiring just minutes of cut-and-paste code editing

Speeding up Existing Mining Algorithms (2/2)

- Time Series Shapelets [39]
  - 18.9 minutes → 12.5 minutes
- Online Time Series Motifs [25]
  - 436 seconds → 156 seconds
- Classification of Historical Musical Scores [10]
  - 142.4 hours → 720.6 minutes (12 hours)
- Classification of Ancient Coins [15]
  - 12.8 seconds per query → 0.8 seconds per query
- Clustering of Star Light Curves [20]
  - 16.57 days → 1.47 days on a single core

Outline

- Introduction
- Explicit Statement of our Assumptions
- Background and Notations
- Known Optimizations
- Novel Optimizations: The UCR Suite
- Experimental Results
- Conclusion

Conclusion

- This work, focusing on fast *sequential search* for DTW, is faster than all known methods s.
- The suite of ideas is 2 to 164 times faster than the state-of-the-art, depending on the query/data.
  - Speed up DTW so that it becomes able to deal with real-time problems.
- However, we are curious about the speed-up ratios of each single idea.

End