



KDD2016

22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining
August 13 - 17, 2016 | San Francisco, California

Extracting Optimal Performance from Dynamic Time Warping

Abdullah Mueen

mueen@unm.edu

Eamonn Keogh

eamonn@cs.ucr.edu

Please cite as: Abdullah Mueen, Eamonn J. Keogh: Extracting Optimal Performance from Dynamic Time Warping. KDD 2016: 2129-2130

Acknowledgements

- The following people helped with advice or images: Diego Silva, Hoang Anh Dau, Tony Bagnall, Mohammad Shokoohi-Yekta, Bing Hu, Chotirat (Ann) Ratanamahatana , François Petitjean (If we forgot anyone, apologies)

- We also acknowledge the following sponsors and funding agencies: MERL Labs, Samsung, Siemens, Google, MSR/IARPA, NSF IIS-1161997 II and SHF-1527127

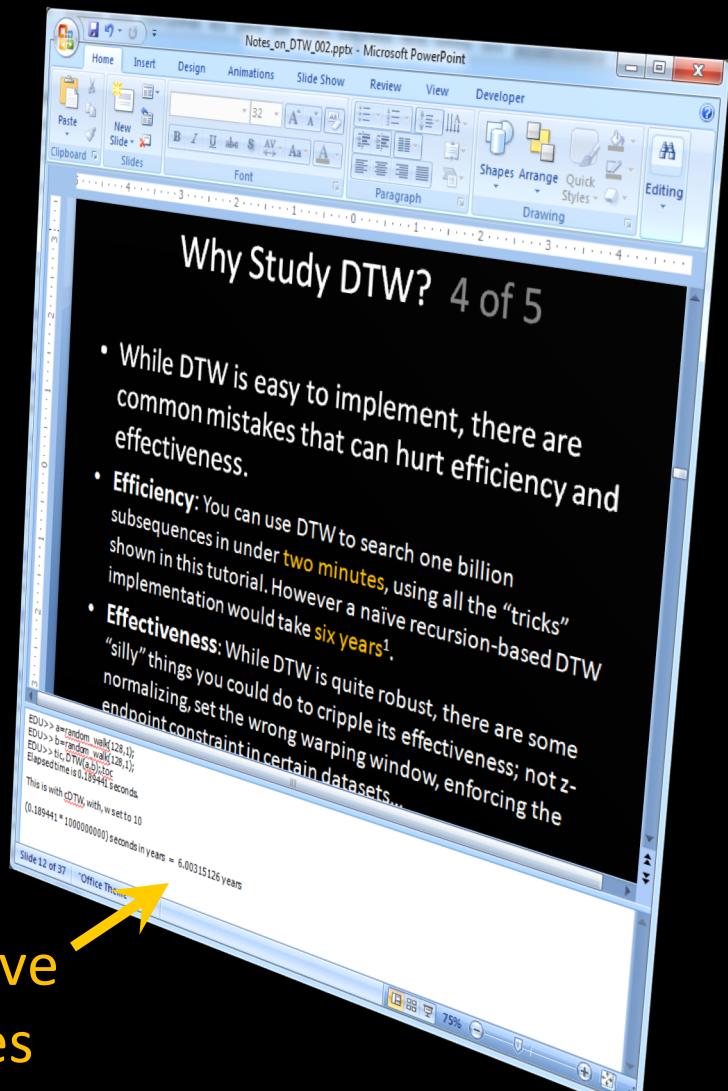


- However any errors or opinions are solely due to A.M and E.K



Disclaimer

- Some of the slides are more “wordy” than we would like.
- Our idea is to expose you to all the important issues in the next few hours. If you need to, you can follow up later by re-reading the slides (augmented by notes/references)
- Feel free to use these slides for teaching, but please let us know if you do so.



Many slides have
additional notes

Approximate Outline

- The First Act: How to do DTW *correctly*

Keogh

- The Second Act: How to do DTW *fast*

Mueen

What are Time Series? 1 of 2

A time series is a collection of observations made sequentially in time.

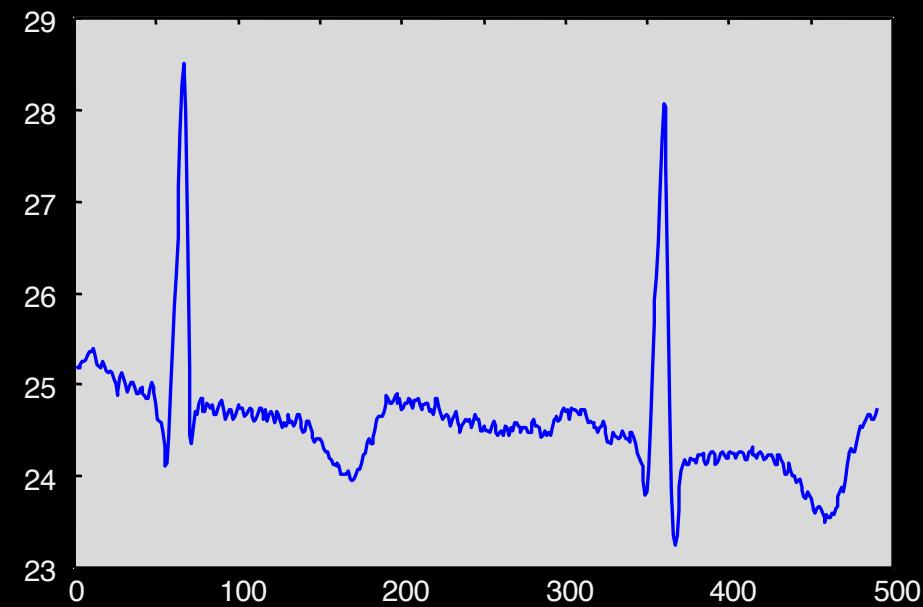
More than most types of data, time series lend themselves to *visual* inspection and intuitions...

For example, looking at the numbers in this blue vector tells us nothing.

But after *plotting* the data, we can recognize a heartbeat, and possibly even diagnose this person's disease.

This tutorial will leverage the visual intuitiveness time series.

25.350
25.350
25.400
25.400
25.325
25.225
25.200
25.175
..
24.625
24.675
24.675
24.675

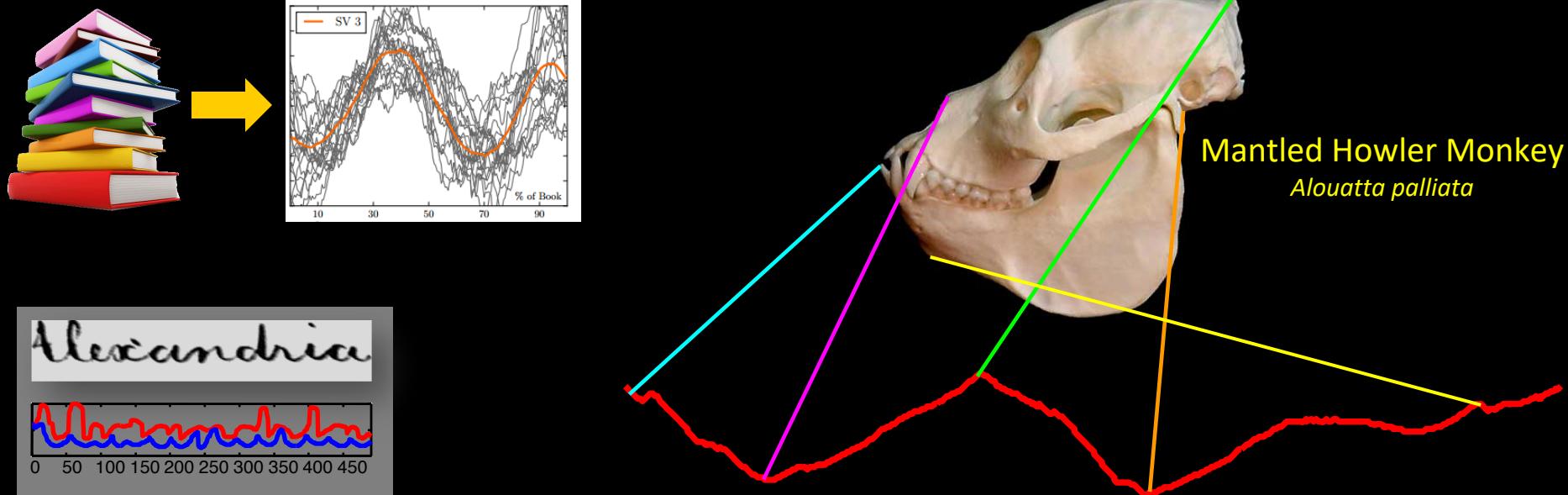


What are Time Series? 2 of 2

As an aside... (not the main point for today)

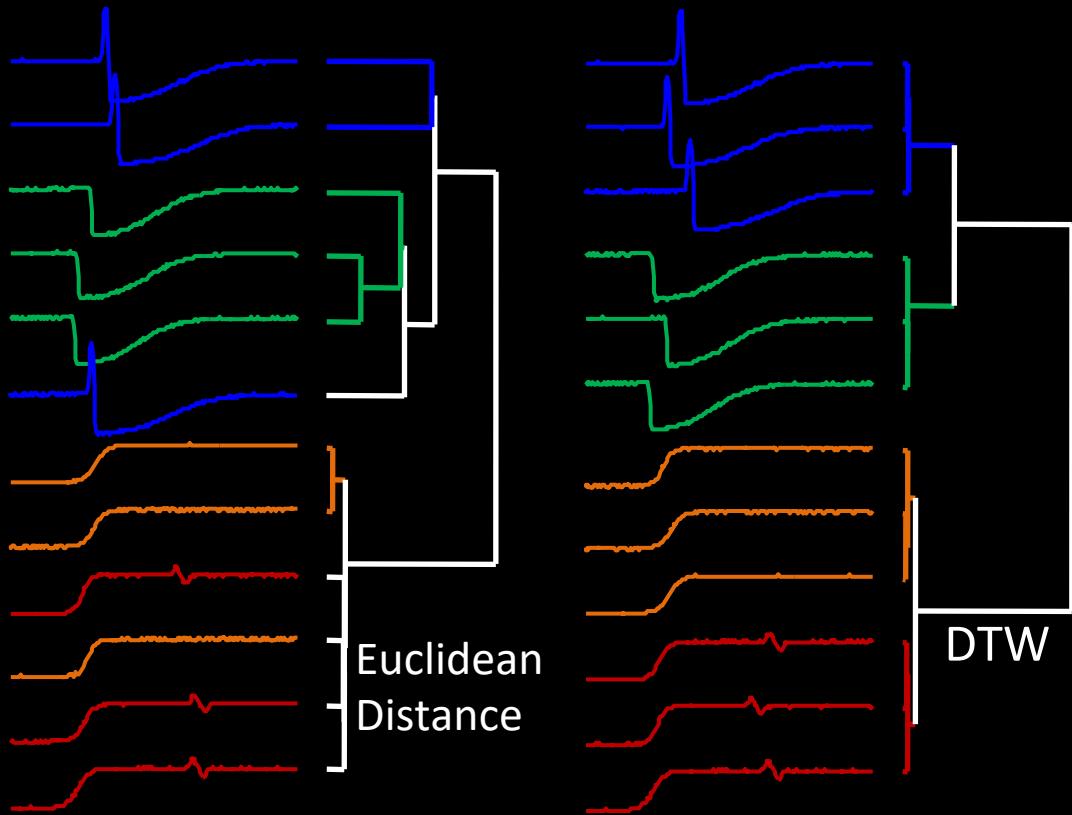
Many types of data that are not *true* time series can be fruitfully transformed into time series, including DNA, speech, textures, core samples, ASCII text, historical handwriting, novels and even *shapes*.

This fact greatly expands the purview of DTW



What is Dynamic Time Warping?

- DTW is an algorithm for measuring similarity between two time series which may vary (i.e. *warp*) in timing.
- This invariance to warping is critical in many domains, for many tasks.
- Without warping invariance, we are often condemned to very poor results.



Why Study DTW? 1 of 5

- Why should you spend several hours studying DTW?
- On the face of it, this *seems* strange.
- We probably would not spend hours on just the Hamming distance, or the just the Jaccard similarity etc.

Why Study DTW? 2 of 5

- It is almost impossible to overstate the ubiquity of DTW in data analytics
- It is used in: robotics, biometrics, medicine, metrology, bioinformatics, video games, gesture recognition, image processing, seismology, music processing, entomology, anthropology, computational photography, bioacoustics, finance ,...

Why Study DTW? 3 of 5

- While there is *no free lunch*, multiple rigorous independent studies¹ show that for the core problem of time series classification, **Nearest Neighbor DTW is very hard to beat.**
- Moreover, where NN-DTW *can* be beaten, it typically by a very small margin, at the cost of huge effort in coding/complexity of implementation, and a large time and space overhead.

¹This paper conducts 35 million experiments, on 85 datasets, with dozens of rival methods
<http://arxiv.org/abs/1602.01711> The Great Time Series Classification Bake Off. Bagnall et al.

Why Study DTW? 4 of 5

- While DTW is easy to implement, there are common mistakes that can hurt efficiency and effectiveness.
- **Efficiency:** You can use DTW to search one billion subsequences in under **two minutes**, using all the “tricks” shown in this tutorial. However a naïve off-the-shelf recursion-based DTW implementation would take **six years**¹.
- **Effectiveness:** While DTW is quite robust, there are some “silly” things you could do to cripple its effectiveness; not z-normalizing, set the wrong warping window, enforcing the endpoint constraint in certain datasets...

¹The recursive version of cDTW₁₀ takes about 0.19 seconds for length 128 (see slide notes)

Why Study DTW? 5 of 5

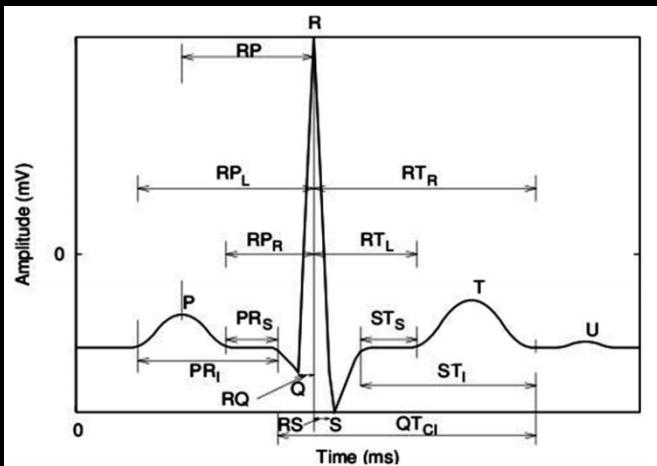
- Finally, many of the ideas we will discuss today to make DTW effective and efficient, can be applied to other domains. (constraining distance measures that are “too” invariant, lower bounding search, just-in-time normalization, early abandoning, cascading multiple lower bounds...)
- So, even if you don’t care about DTW per se, we hope you will find some ideas you can use.

An Unstated Assumption

For all time series, faces, gait, fish and heartbeats, we could design and extract **features**, and just represent the objects as **feature vectors**. However, in the dozens of cases where we compared this type of approach to just using DTW on the raw data, DTW almost always wins!

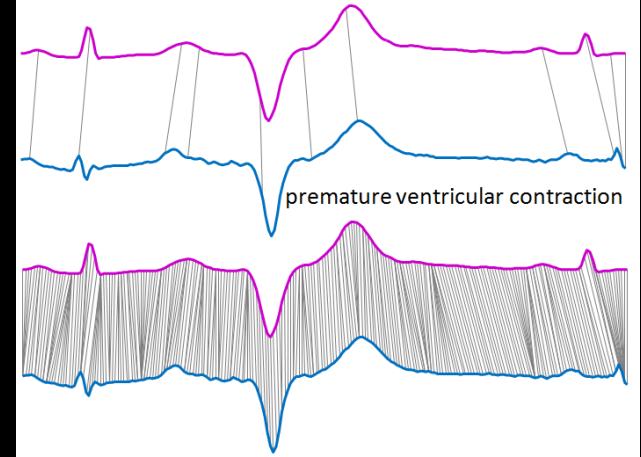
So, at the very least, try the simplest ideas first, and DTW is very simple.

Bazett's formula



Accuracy
75 to 95%
(10 classes)

1NN DTW
Accuracy
98% plus
(10 classes)



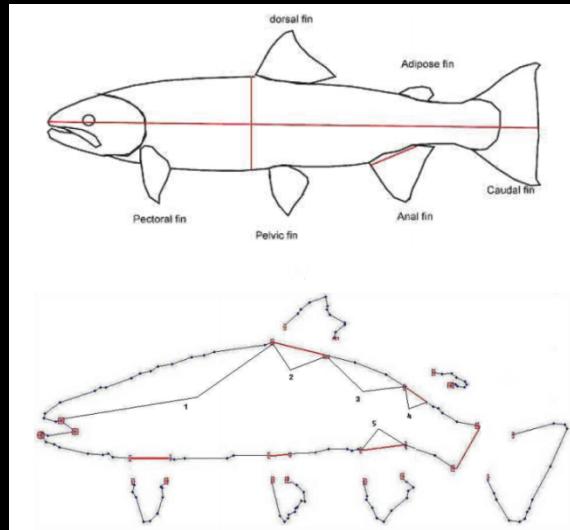
PR Interval, PR Segment, Corrected-QT Interval, ST Segment , ST Interval , RR Interval , RQ Amplitude, R_{peak} to T_{onset} Segment RS Amplitude , Angle Q, Angle R, Angle S,...
 $F1=\{7.3, 4.2, 5.2, 1.2, 6.7, \dots\}$

Parameter-free or parameter-lite,
robust to noise etc.

An Unstated Assumption (alterative)

For all time series, heartbeats, gait, faces, and even fish!, we could design and extract **features**, and just represent the objects as **feature vectors**. However, in the dozen of cases where we compared this type of approach to just using DTW on the raw data, DTW almost always wins!

So, at the very least, try the simplest ideas first, and DTW is very simple.



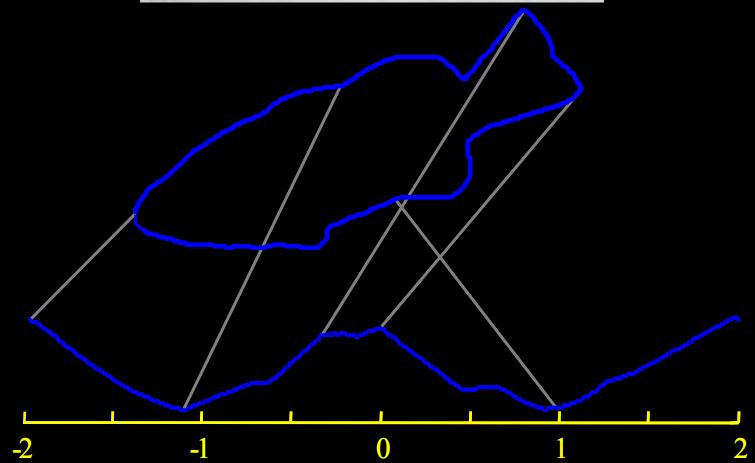
Accuracy
75.7%

1NN DTW
Accuracy
86.0%



1. length between the nose and dorsal fin
2. width of the dorsal fin
3. distance between the dorsal fin and adipose fin
4. width of the adipose fin
5. width of the anal fin

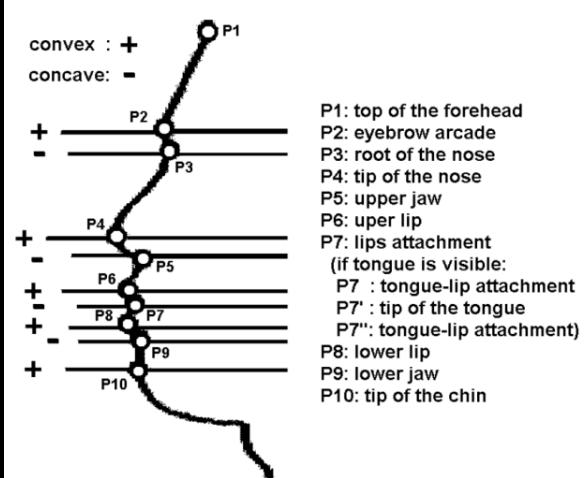
$$F1=\{5.1, 1.2, 2.9, 1.0, 2.2\}$$



An Unstated Assumption (alterative)

For all time series, heartbeats, gait, fish and faces, we could design and extract **features**, and just represent the objects as **feature vectors**. However, in the dozen of cases where we compared this type of approach to just using DTW on the raw data, DTW almost always wins!

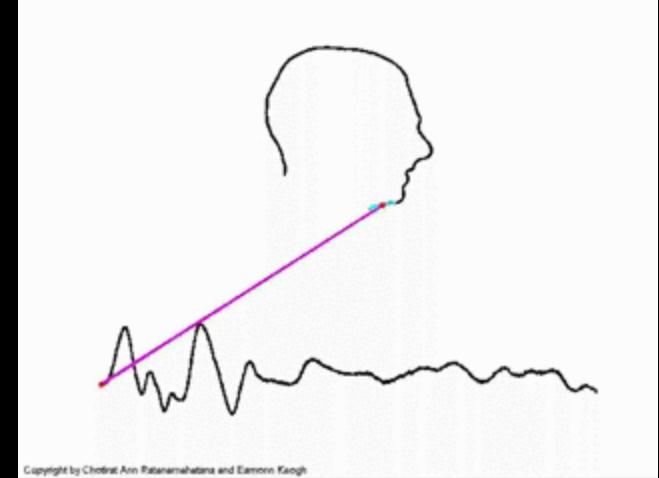
So, at the very least, try the simplest ideas first, and DTW is very simple.



How many fiducials? Pantic suggests 10, Campos suggests 8, Dariush suggests 9, Liposcak suggests 12...

Accuracy
70 to 80%
(10 classes)

1NN DTW
Accuracy
90% plus
(10 classes)



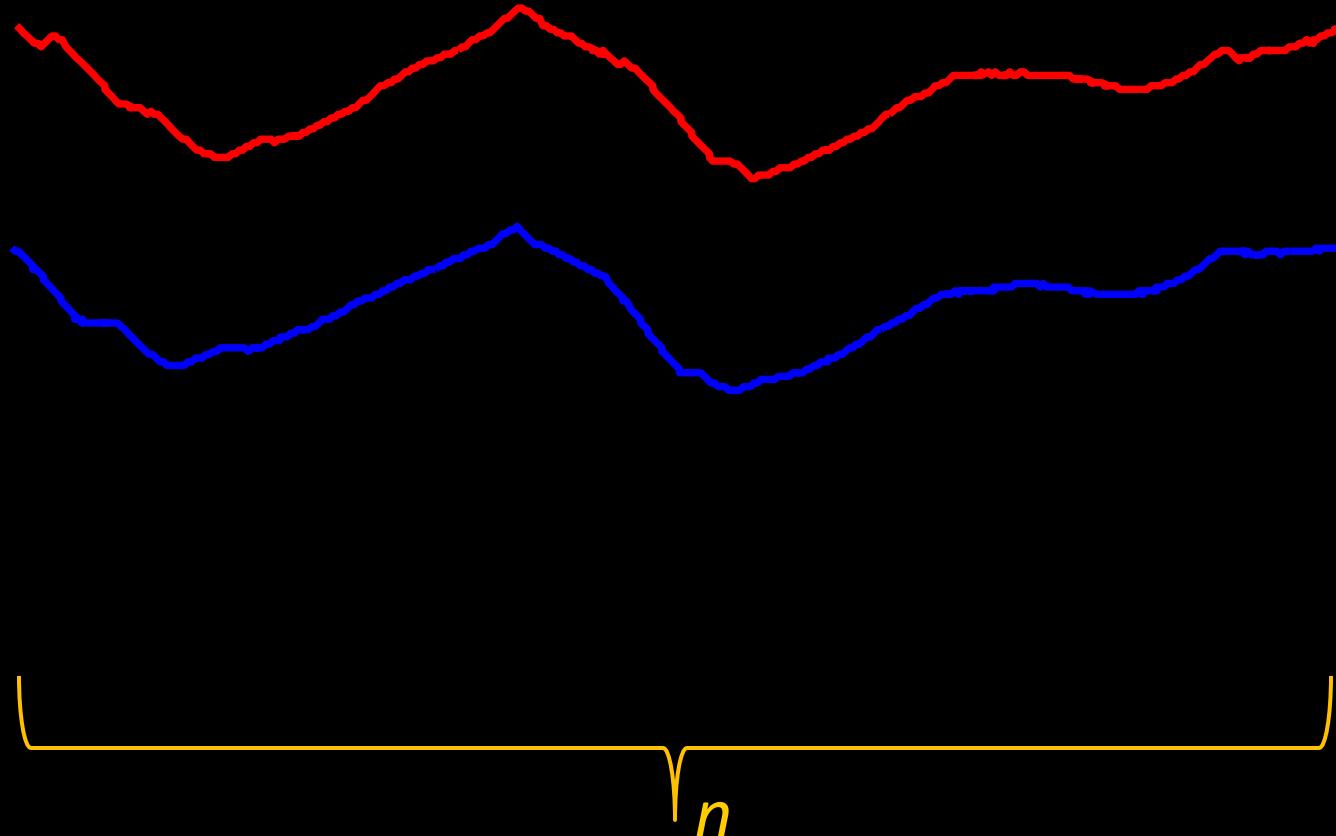
Copyright by Chedrat Ann Ratenarathana and Eamonn Keogh

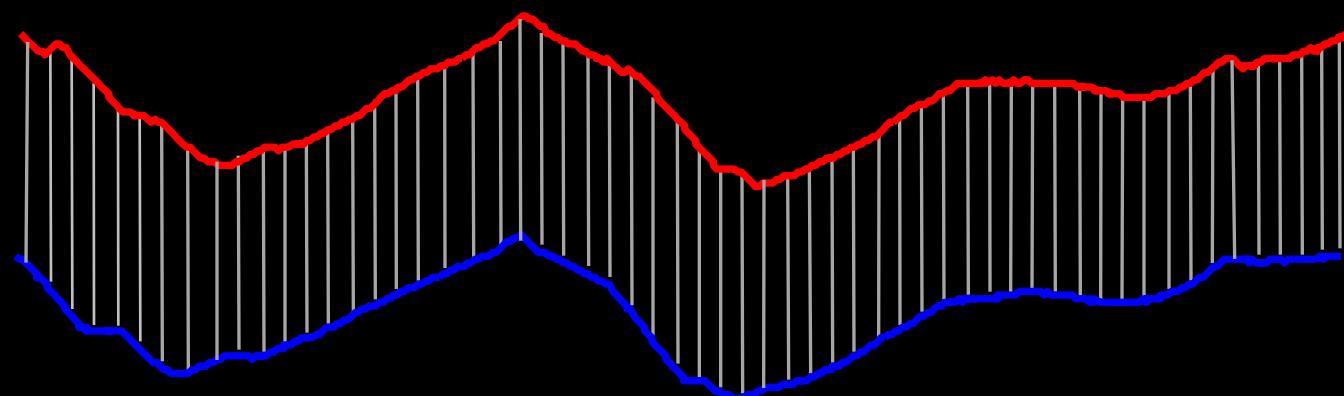
Parameter-free or parameter-lite,
robust to changes of expression..

$$F1=\{7.4, 1.3, 2.1, 1.2, 4.6, 5.6, 43.3\}$$

A Visual Intuition of Distance Measures

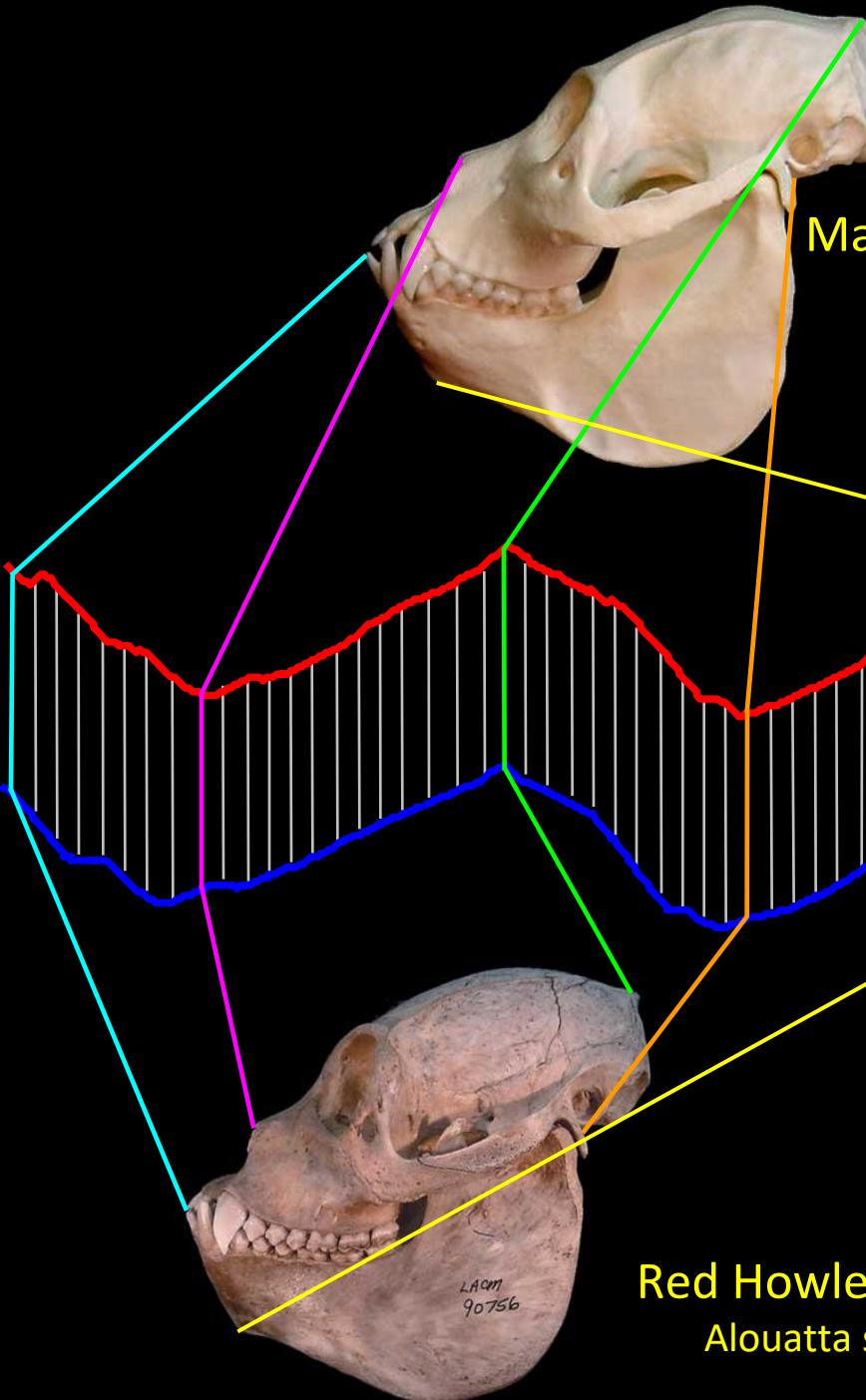
We have two time series, what is the distance between them?
Equivalently, how similar are they?





Euclidean
Distance

One-to-one mapping

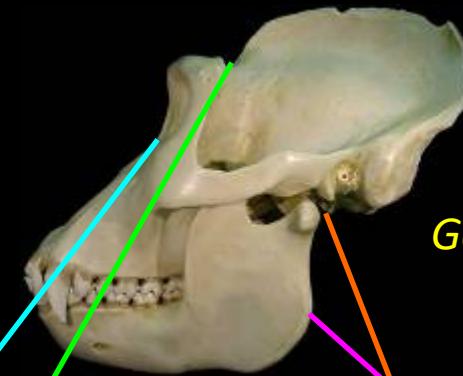


Mantled Howler Monkey
Alouatta palliata

Euclidean
Distance

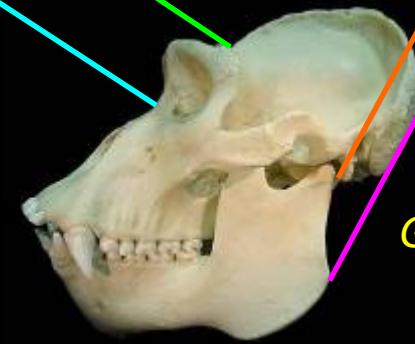
Red Howler Monkey
Alouatta seniculus

Lowland Gorilla
Gorilla gorilla graueri

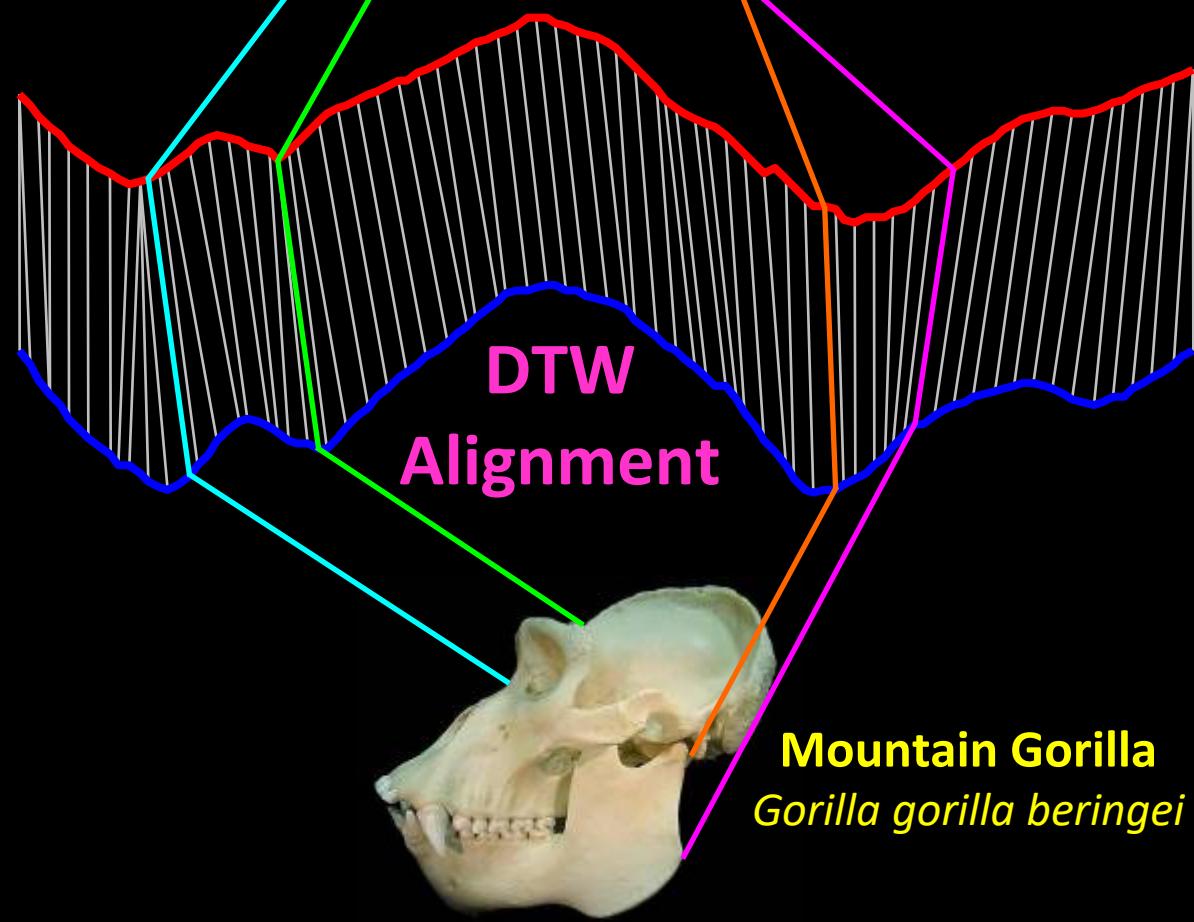


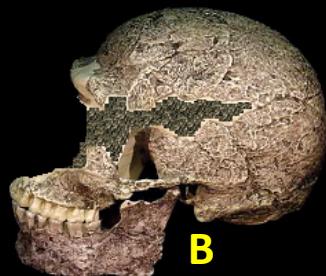
**DTW
Alignment**

Mountain Gorilla
Gorilla gorilla beringei



One-to-many mapping

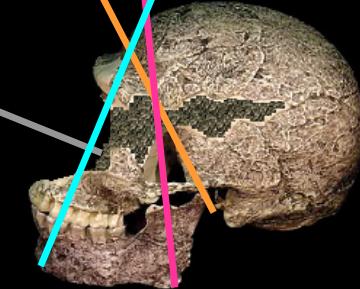
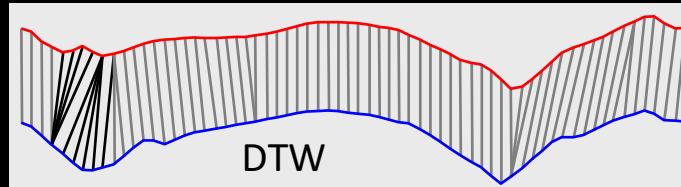




This region
will not be
matched

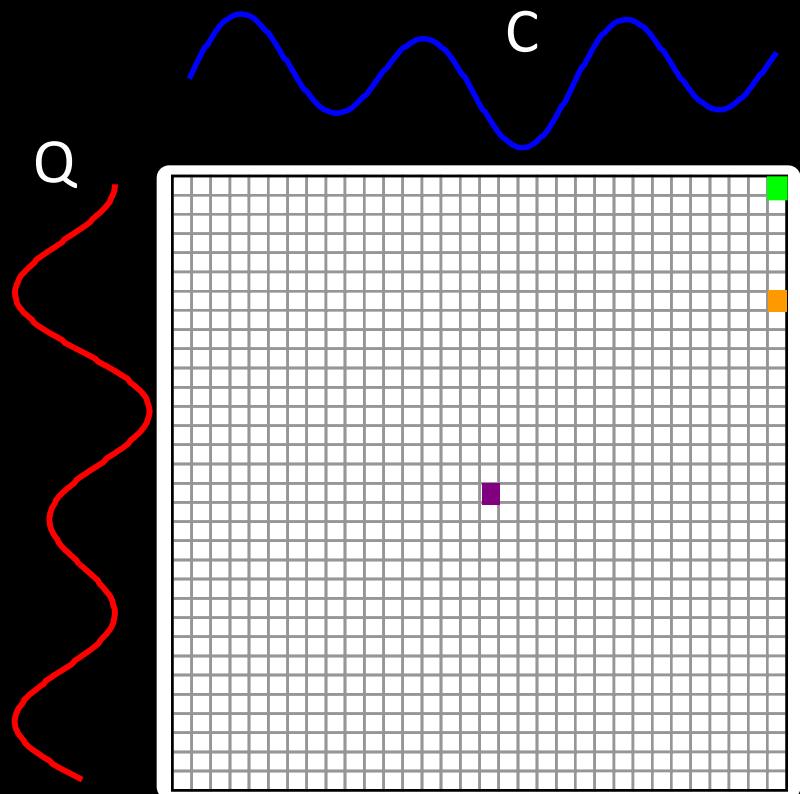
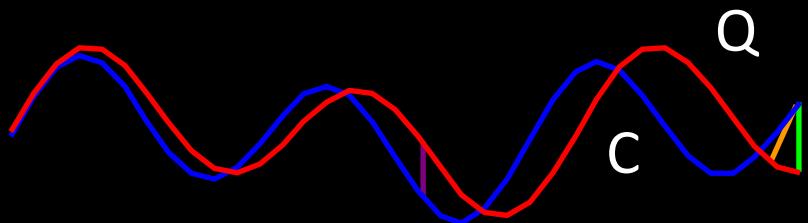
LCSS Alignment

One-to-many mapping
With some points allowed
to be unmapped



How is DTW Calculated? I

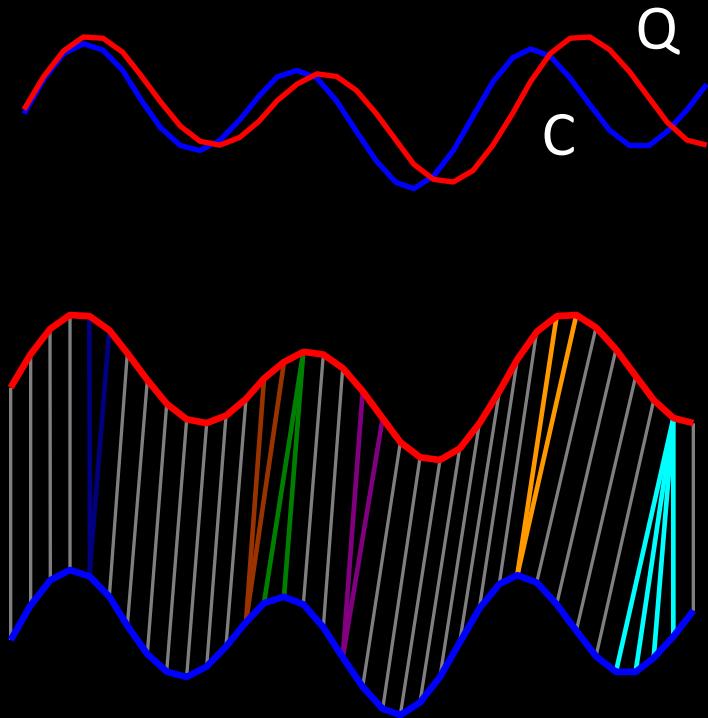
We create a matrix the size of $|Q|$ by $|C|$, then fill it in with the distance between every possible pair of points in our two time series.



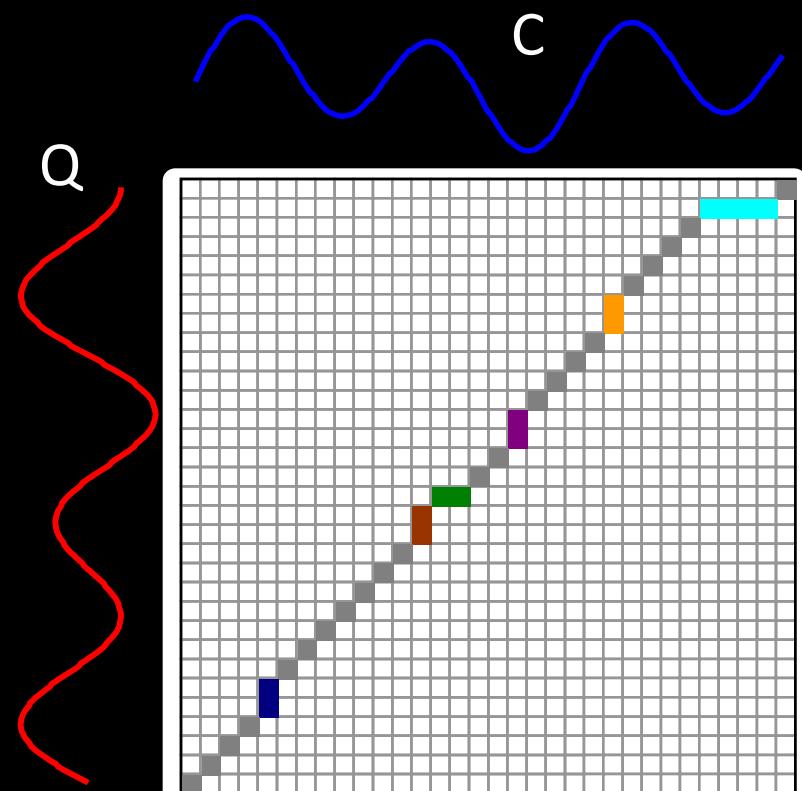
How is DTW Calculated? II

Every possible warping between two time series, is a path through the matrix. We want the *best* one...

$$DTW(Q, C) = \min \left\{ \sqrt{\sum_{k=1}^K w_k} \right\} / K$$



$$\gamma(i,j) = d(q_i, c_j) + \min\{ \gamma(i-1, j-1), \gamma(i-1, j), \gamma(i, j-1) \}$$



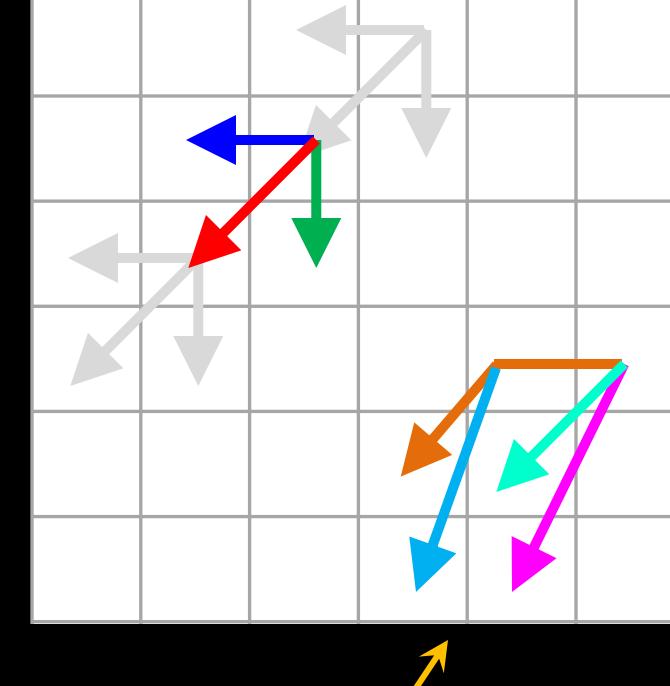
Warping path w

We can visualize the recursive function as a “step pattern” of allowable moves, or search operators

$$\gamma(i,j) = d(q_i, c_j) + \min\{ \gamma(i-1, j-1), \gamma(i-1, j), \gamma(i, j-1) \}$$

This suggests two generalizations.

- We could weight the diagonal step, to “discourage” the warping path wandering too far from the diagonal.
- We could create other steps patterns, including asymmetric step patterns.



RabinerJuangStepPattern IV

Both ideas were extensively studied when DTW was the dominant speech processing algorithm, but have not been investigated extensively in the data mining context.

Empirically, they seem to make little difference.

We only consider the classic **symmetric1** step pattern in our work.

How is DTW Calculated? *Disclaimer!*

In practice we **don't** use *recursion* to calculate DTW. Instead we use an equivalent *iterative* method (which we will explain later).

The iterative method is both absolutely faster, and it allows many speed-up optimizations (early abandoning etc).

The time difference is several orders of magnitude.

However, there is no logical difference between the two methods for the first half of this tutorial.

This recursive function gives us the minimum cost path

$$\gamma(i,j) = d(q_i, c_j) + \min\{ \gamma(i-1, j-1), \gamma(i-1, j), \gamma(i, j-1) \}$$



Logically correct, but too slow and memory intensive

DTW is a Distance *Measure*, not a *Metric* 1 of 2

Requirements to be a metric

Yes for DTW

$$D(A,B) = D(B,A)$$

Symmetry

$$D(A,A) = 0$$

Constancy of Self-Similarity

No for DTW

$$D(A,B) = 0 \text{ IIf } A=B$$

Positivity (Separation)

$$D(A,B) \leq D(A,C) + D(B,C)$$

Triangular Inequality

Normally we prefer metrics over measures for two reasons:

- Non-Metrics can sometimes give pathological solutions when clustering or classifying data etc.
- Almost all speed-up “tricks” for high dimensional data exploit the Triangular Inequality.

DTW is a Distance *Measure*, not a *Metric* 2 of 2

- Non-Metrics can sometimes give pathological solutions when clustering or classifying data etc.

DTW is *almost* a metric! That is to say if you randomly sample a million triples A, B and C, you will probably find $D(A,B) \leq D(A,C) + D(B,C)$ is obeyed for 999,999 of them.

*Moreover, in the limit, as w approaches zero, DTW is a metric. As we will see, we almost always use a very small value of w .

- Almost all speed-up “tricks” for high dimensional data exploit the Triangular Inequality.

This used to be a big issue, but after a decade of research, the community has found alternative techniques that allow us to speed up DTW (Second half of the tutorial).

*We will explain what “ w ” is in a few slides.

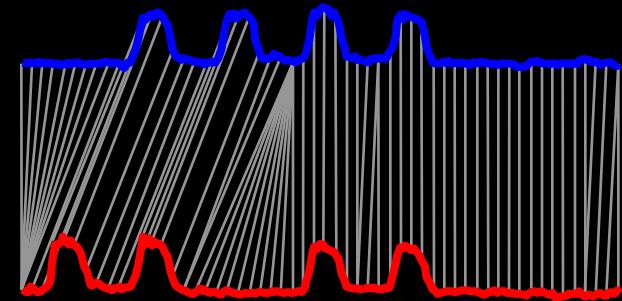
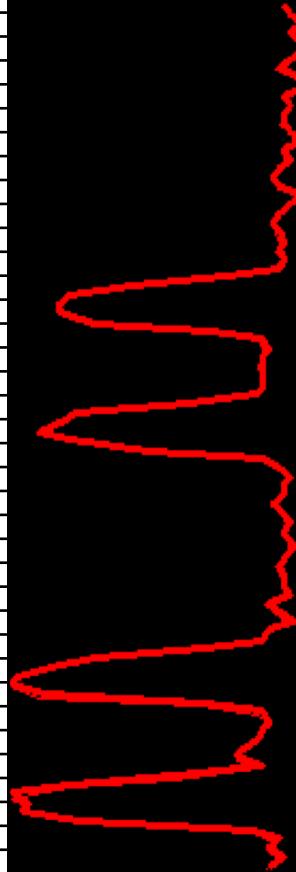
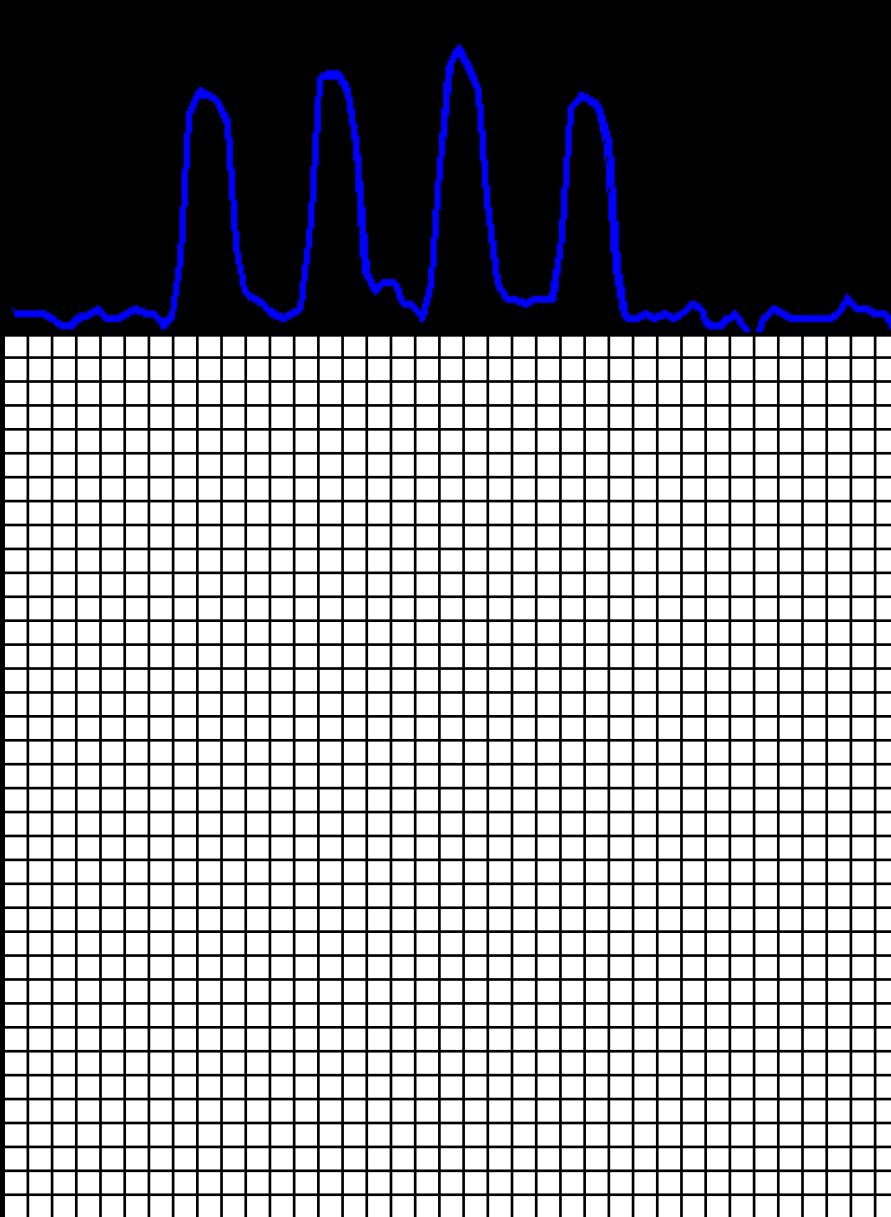
DTW: Time and Space complexity

- The “off-the-shelf” DTW has
- a time complexity of $O(n^2)$ (with a large constant factor)
- a space complexity of $O(n^2)$

This is the most cited reason for not using DTW.

- However, as we will show in the second half of this tutorial. DTW can have
 - a space complexity of $O(n)$
 - an amortized time complexity of $O(n)$ (with a very small constant factor)

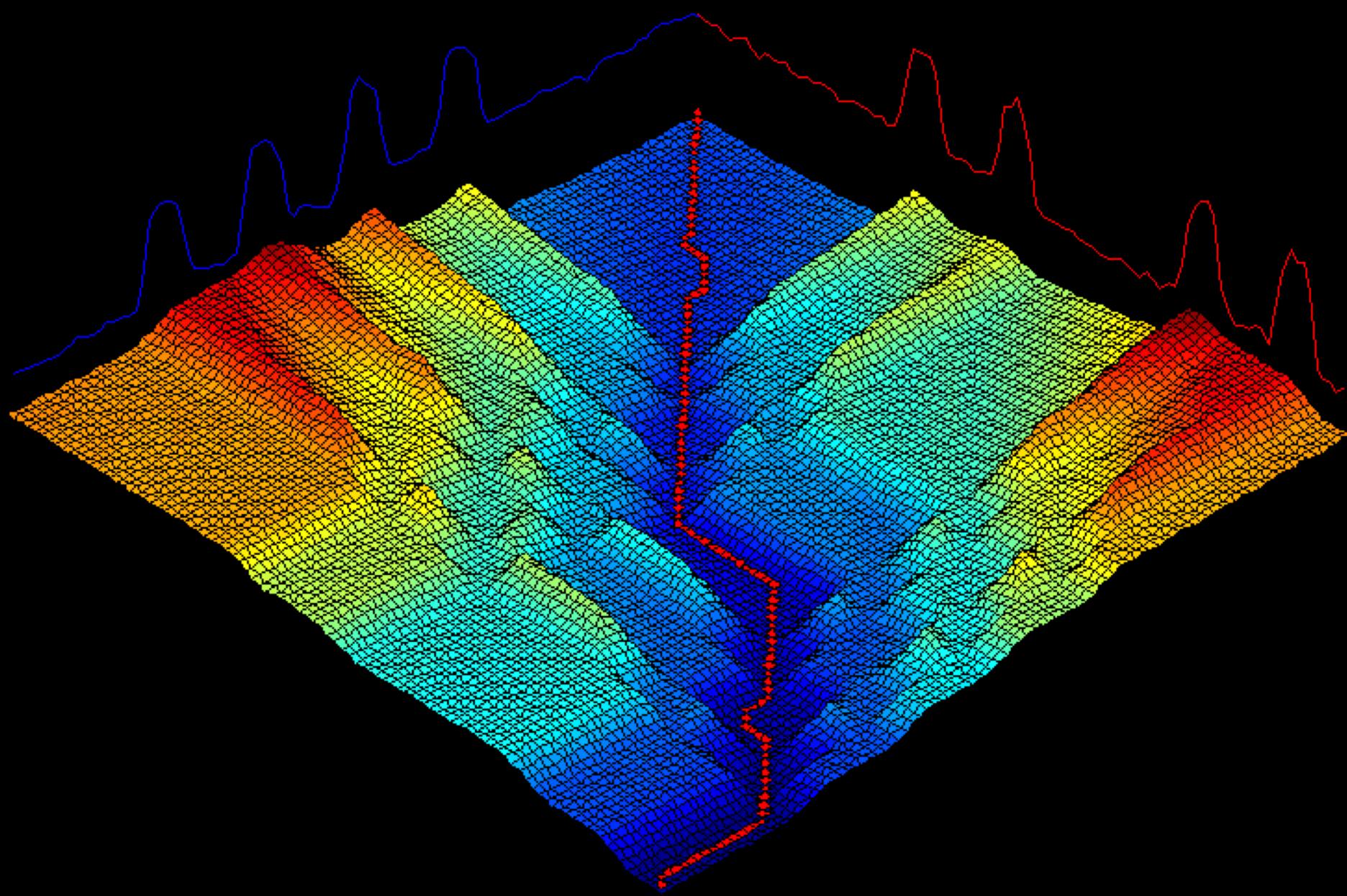
Let us visualize the cumulative matrix on a real world problem I



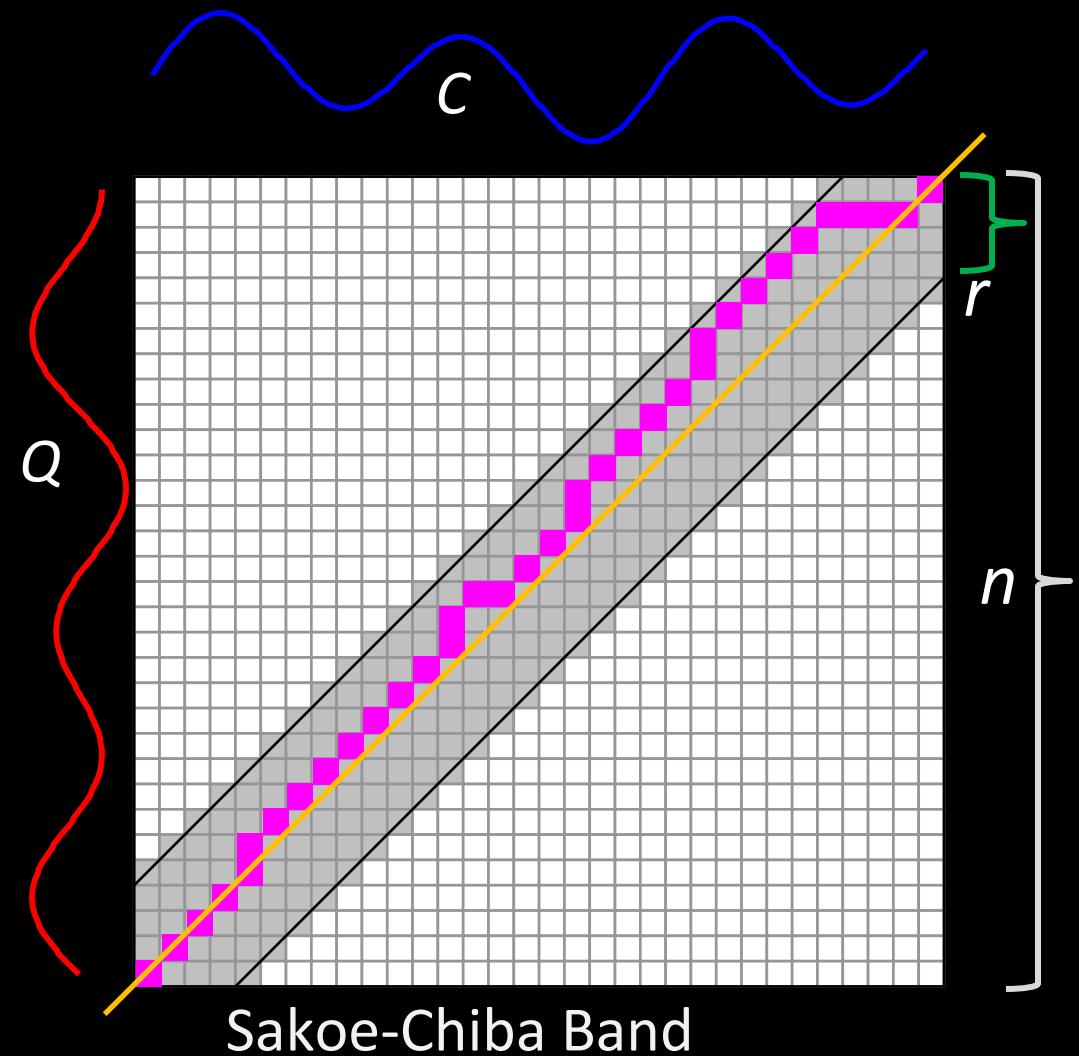
This example shows two one-week periods from an electrical power demand time series.

Note that although they both describe 4-day work weeks, the blue sequence had Monday as a holiday, and the red sequence had Wednesday as a holiday.

Let us visualize the cumulative matrix on a real world problem II



Understanding w , the Warping Constraint



The value of w is the maximum amount the warping path is allowed to **deviate** from the **diagonal**.

It is normally expressed as the ratio
 $w = r/n$ (or as a percentage)

We need to understand w because:

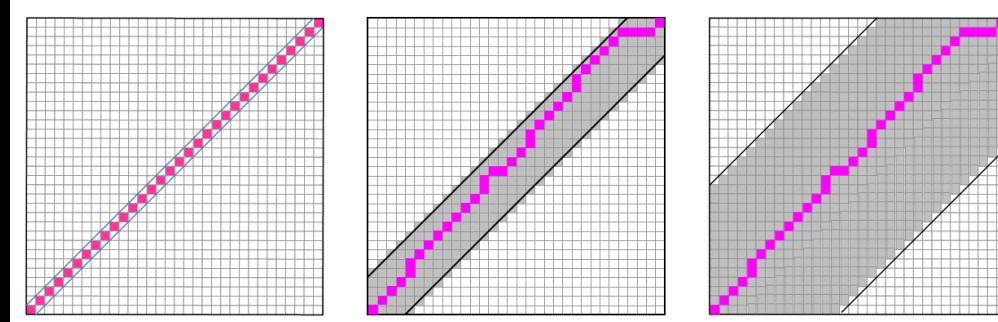
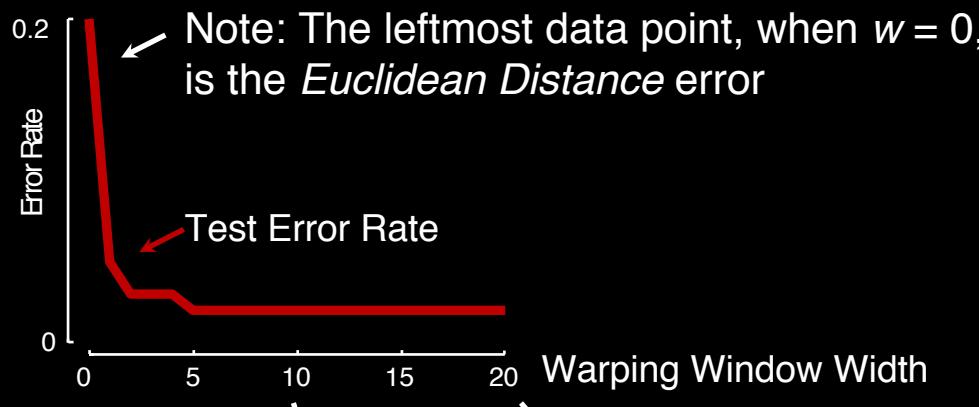
- The most useful speedup tricks all exploit w .
- The value chosen for w can greatly affect accuracy.

Preamble: Reading the plots for..

The value of w vs. some measure of quality

Testing every value of w , from $w = 0$ to $w = 20$

This is just a meta-slide, on how to read a figure

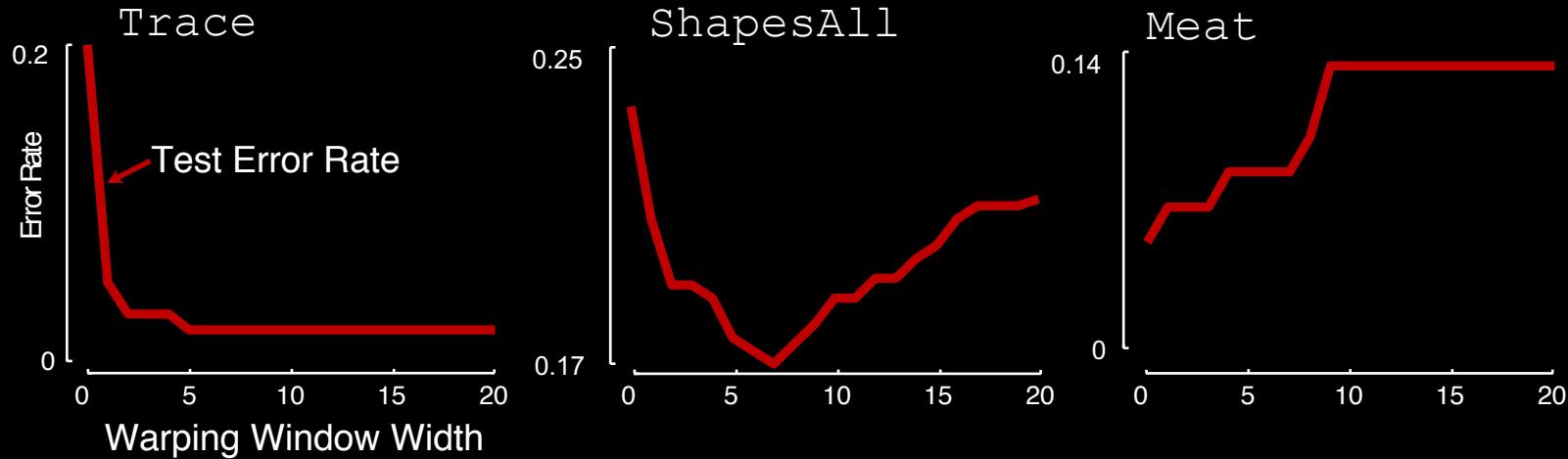


We use the simple 1NN classifier

Time needed: Milliseconds,... Seconds,... Minutes,... (at least, if naively implemented)

How important is the value of w ?

To find out, we measured the testing accuracy on some UCR datasets
We tested every value of w , from $w = 0$ to $w = 20$

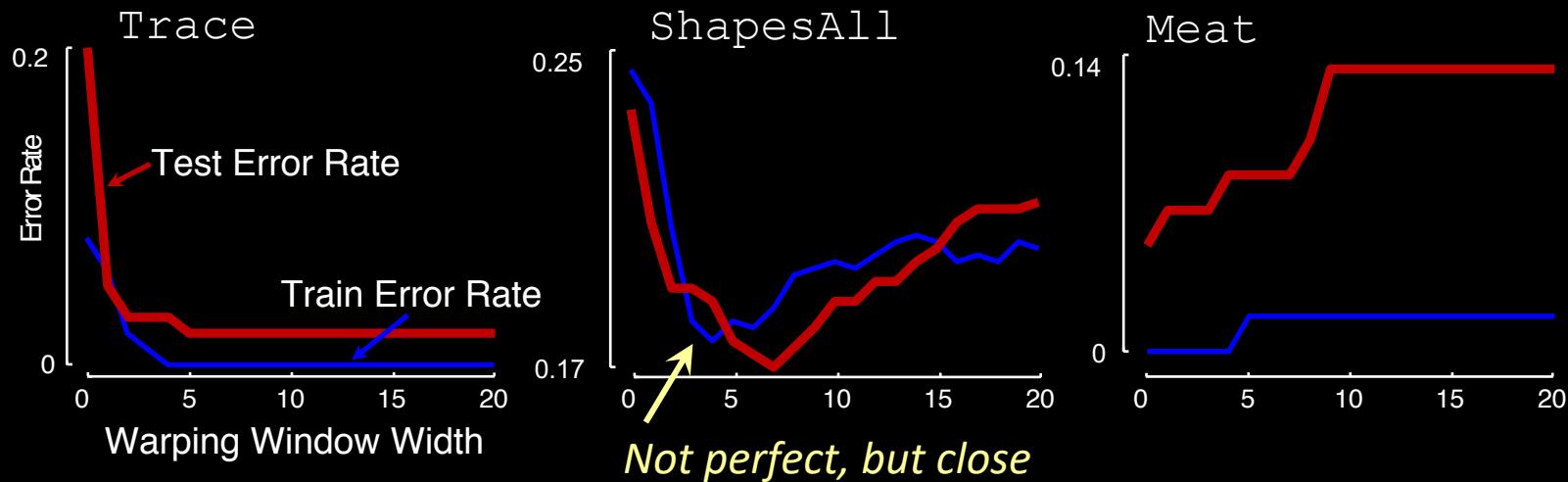


There is high variability: Any warping *hurts* for Meat (such datasets are rare), any amount of warping above 5 is optimal for Trace, and ShapesAll really needs 7, not more, not less.

How can we set the value of w ?

How can we set the value of w ?

We can set the value of w using leave-one-out cross-validation on the testing data. So long as we have enough labeled data, this generally works very well, as shown below:



What do you do if you don't have enough labeled data? (open problem)
One Idea: Find a dataset that is similar, that *does* have labeled data, and hope the best setting generalizes from that dataset.

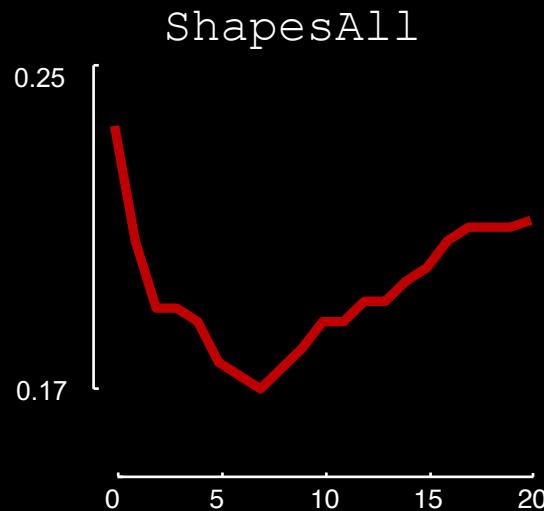
The value of w vs. data size

For classification and query-by-content, the best setting of w also depends on the *size* of the training dataset.

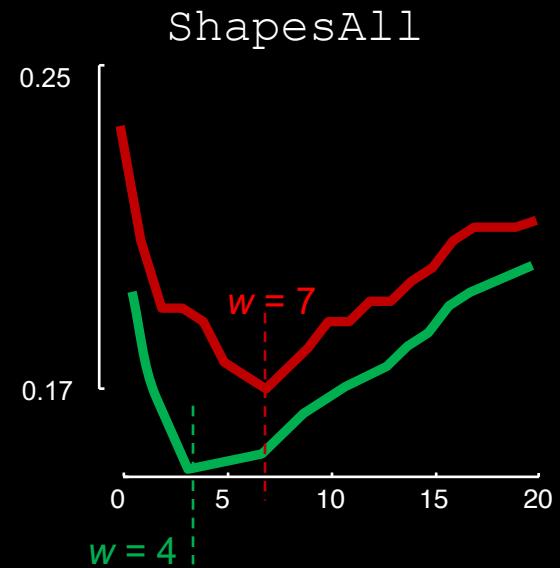
If we are given more training data, we should expect:

- The error rate to decrease (as with all ML problems and all data types)
- The best value for w to get smaller (it is possible to construct synthetic counterexamples)

This implies that with enough training data, you would not need DTW!



If given more
training data....
→
... we expect the
best value for w to
decline.



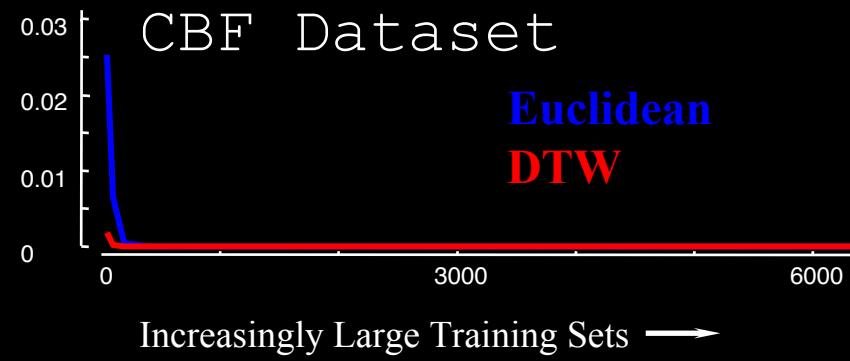
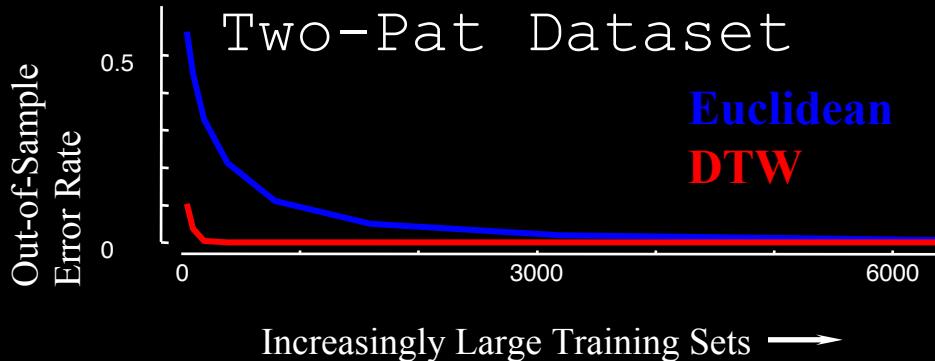
..this implies that with enough training data, you would not need DTW!

Empirically, this seems to be universally true. 1NN-DTW classification is generally much more accurate than 1NN-ED classification for small datasets. However, as you add more training data, the gap begins to close, and eventually converges.

To our knowledge, there is no research on quantifying how fast they converge, if they must converge, the relative benefit of a new training object for each approach etc.

As a practical matter, this observation probably does not matter much.

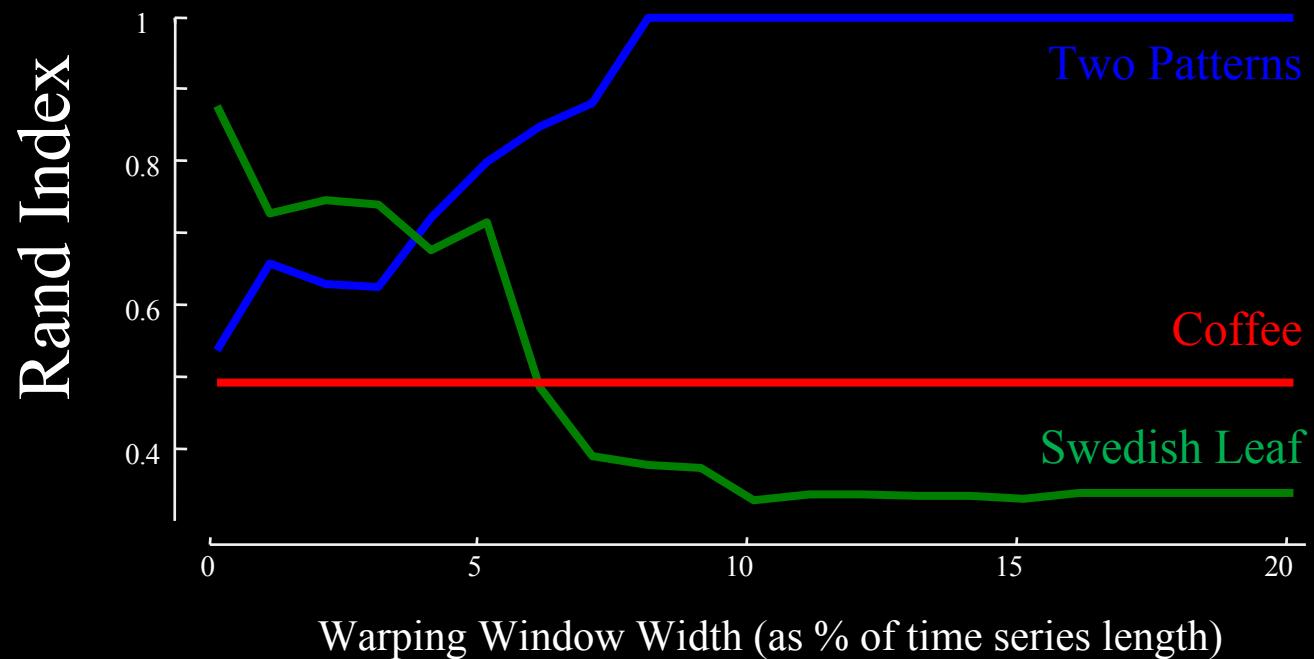
- We rarely have the luxury lots of training data
- If we did, it would probably still be faster to do 1NN-DTW with less data, than 1NN-ED with lots of data.



Revisited: How important is the value of w ? 1 of 5

While the community has just begun to understand that the setting of w is important for ***classification***, it is not well appreciated that it is just as important for ***clustering***.

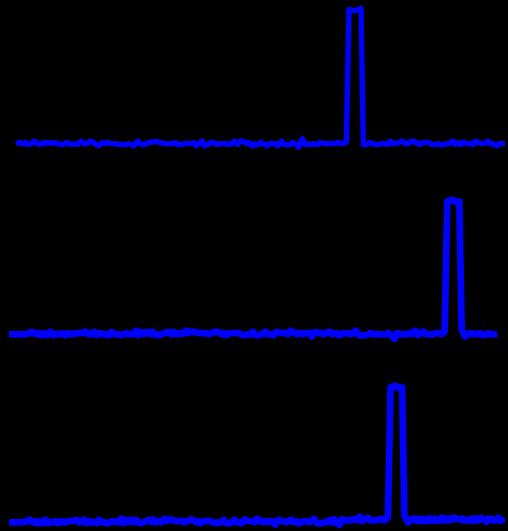
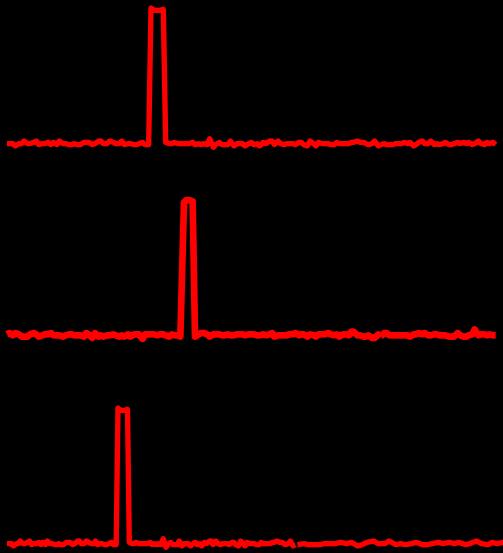
Below we show the rand-index vs w for three datasets, clustered using TADPole. As we can see, the value of w is critical.



Revisited: How important is the value of w ? 2 of 5

Why is the value of w so critical? The Goldilocks Principle

Lets *make* a dataset of time series of length 100. The dataset is mostly a flat line with a little noise. The red class has a single “spike” somewhere between 1 and 40. The blue class has a single “spike” somewhere between 60 and 100.

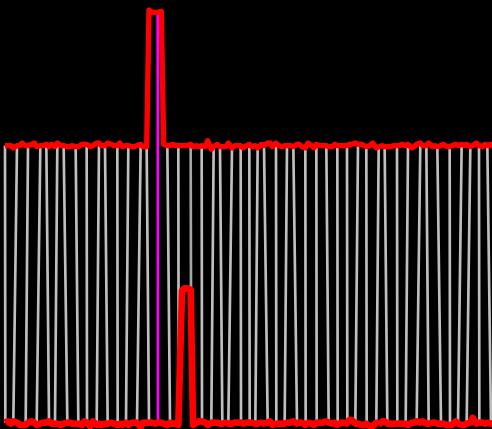


Revisited: How important is the value of w ? 3 of 5

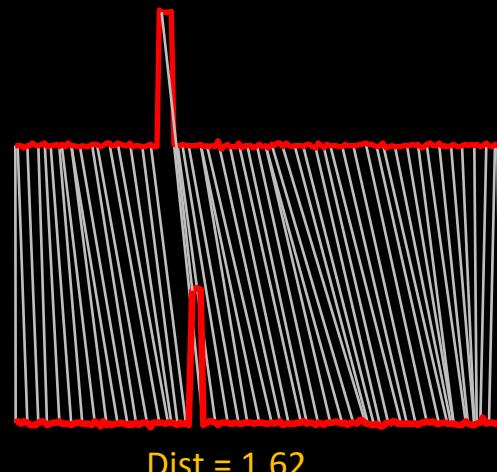
Why is the value of w so critical? The Goldilocks Principle

Lets *make* a dataset of time series of length 100. The dataset is mostly a flat line with a little noise. The red class has a single “spike” somewhere between 1 and 40. The blue class has a single “spike” somewhere between 60 and 100.

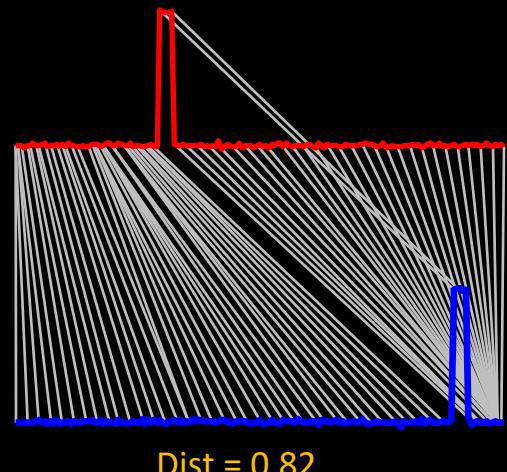
With *no* warping allowed,
we map the top of the spike
on the source to a low spot
on the target.



With *a little* warping
allowed, we map the spike
on the source to spike on
the target. Success!



With *unlimited* warping
allowed, we can map the
top of the spike on the
source to a very far away
spike, which what defines
the opposite class!

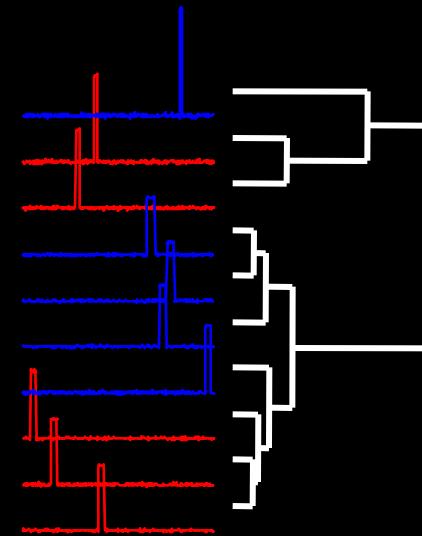
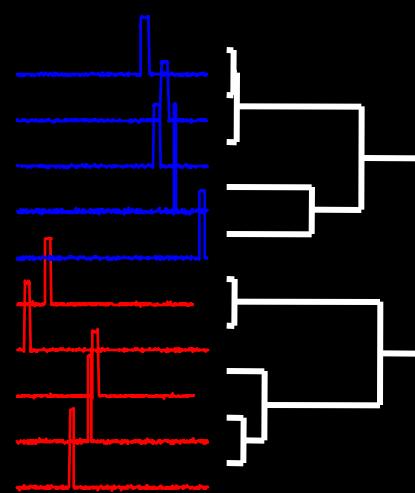
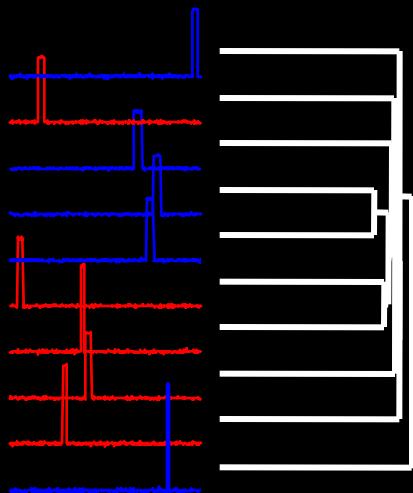


Revisited: How important is the value of w ? 4 of 5

Why is the value of w so critical? The Goldilocks Principle

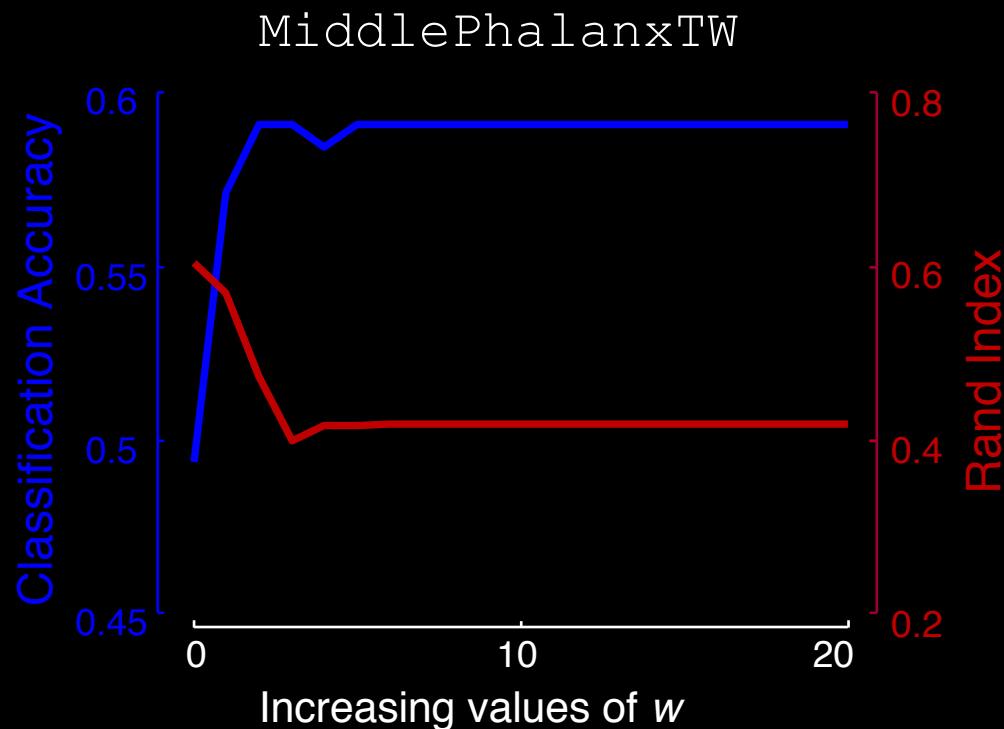
Lets *make* a dataset of time series of length 100. The dataset is mostly a flat line with a little noise. The red class has a single “spike” somewhere between 1 and 40. The blue class has a single “spike” somewhere between 60 and 100.

- With no warping allowed, we get random results
- With a little warping allowed (say 10%), a red can “warp” to a red, and vice versa
- With 100% warping allowed, a blue could warp to a red, and vice versa.



Revisited: How important is the value of w ? 5 of 5

Important note: The best value of w for **classification** may not be the best value for **clustering**.



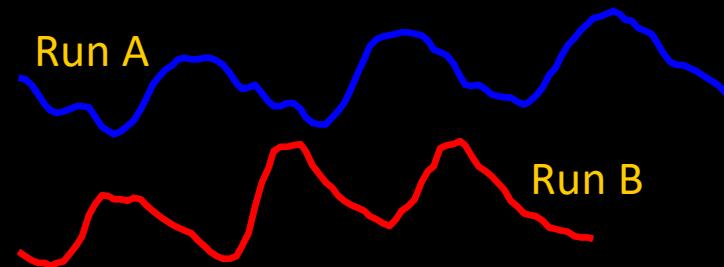
In retrospect it is not surprising that these values are at best weakly related. For 1NN classification only the distance between the unlabeled exemplar and its *single* nearest neighbor matters.

However, for clustering, the mutual distance among small *groups* of objects matter.

The **Rand-Index** and the **classification accuracy** vs. the warping window width, for the MiddlePhalanxTW dataset.

Time series of different lengths 1 of 2

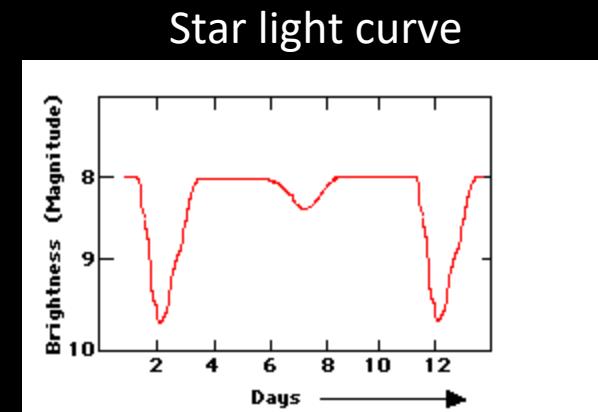
What if you have two time series that are different lengths? For example, here Run B is only 81% the length of Run A.



This situation occurs quite a lot:

- Heartbeats recorded at 60bpm vs. 50bpm
- Star light curves come in discrete classes, but each class can have variable periods.
- An utterance of '*Mississippi*' may take 0.6 to 0.9 seconds
- Etc.

This *global* difference in scaling is typically independent of the *local* scaling that DTW is designed to handle.



Time series of different lengths 2 of 2

What if you have two time series that are different lengths? For example, here B is only **81%** the length of A.

There are at least two things you can do:

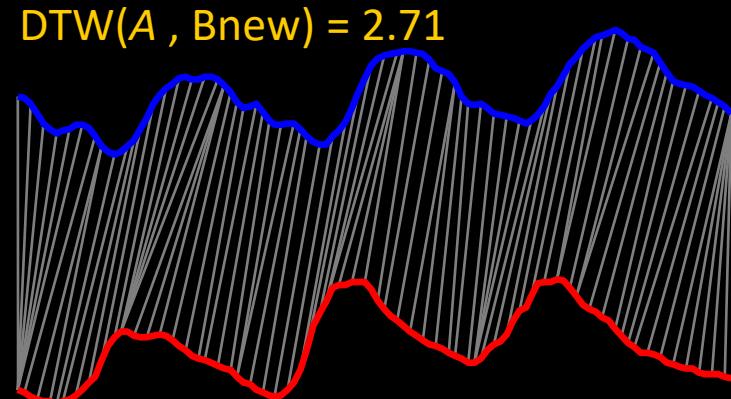
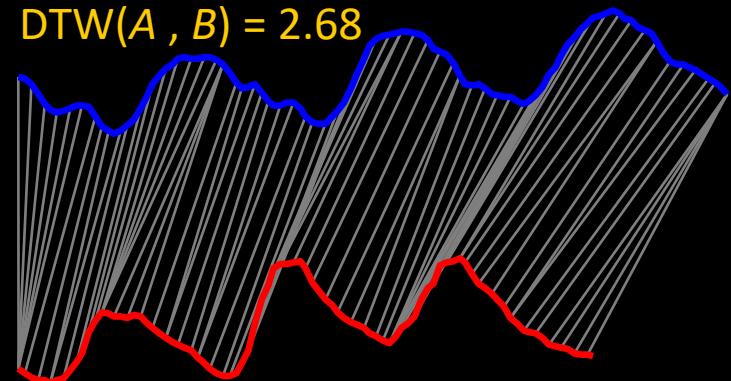
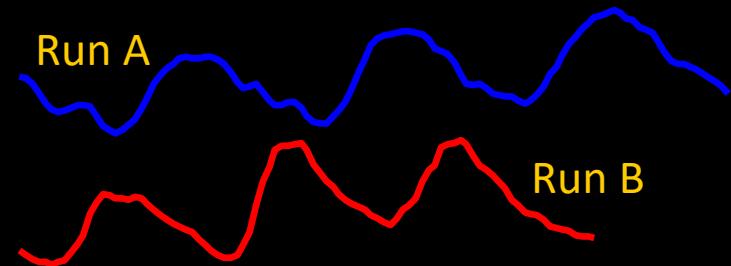
- 1) Compare them in their native lengths.
- 2) Re-interpolate them to have the same length.

An inelegant, but quick way to do this in Matlab is:

```
Bnew= B(1:0.81:end);
```

Empirically makes little or no difference for classification, clustering etc.

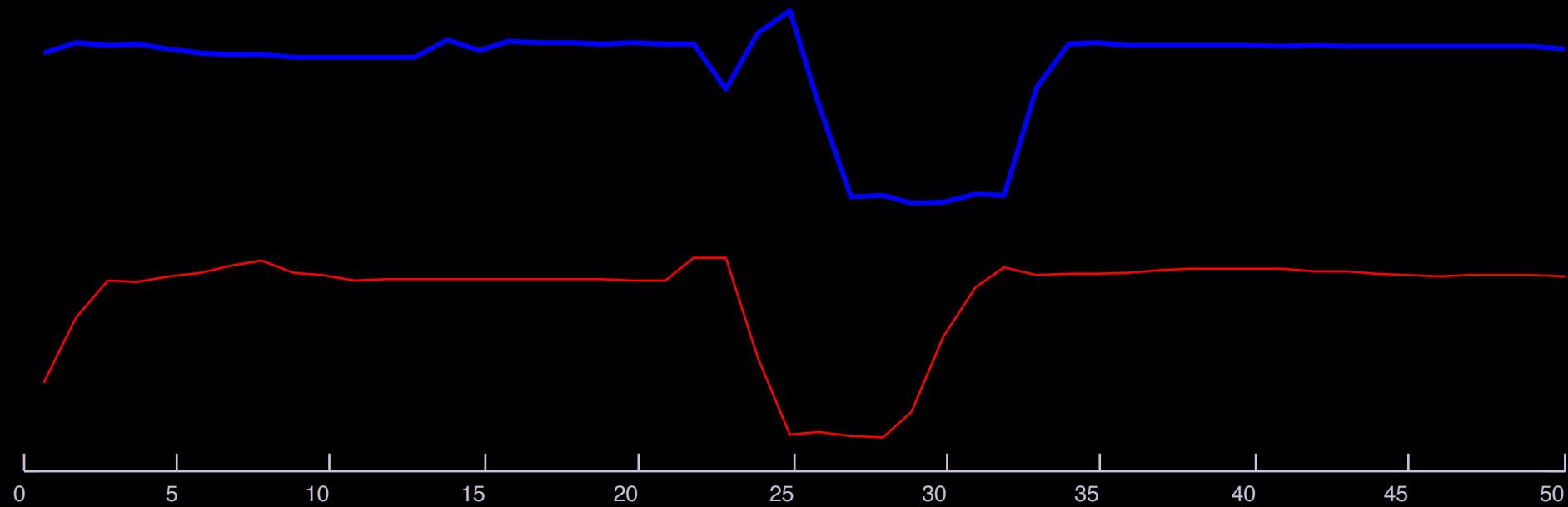
However, the equal length case is much easier to handle, so we assume it from now on.



The Importance of Endpoints: 1 of 3

Consider the two time series below. They are similar, but out of phase, *exactly* the case that DTW is designed for.

Let us compare them using DTW....



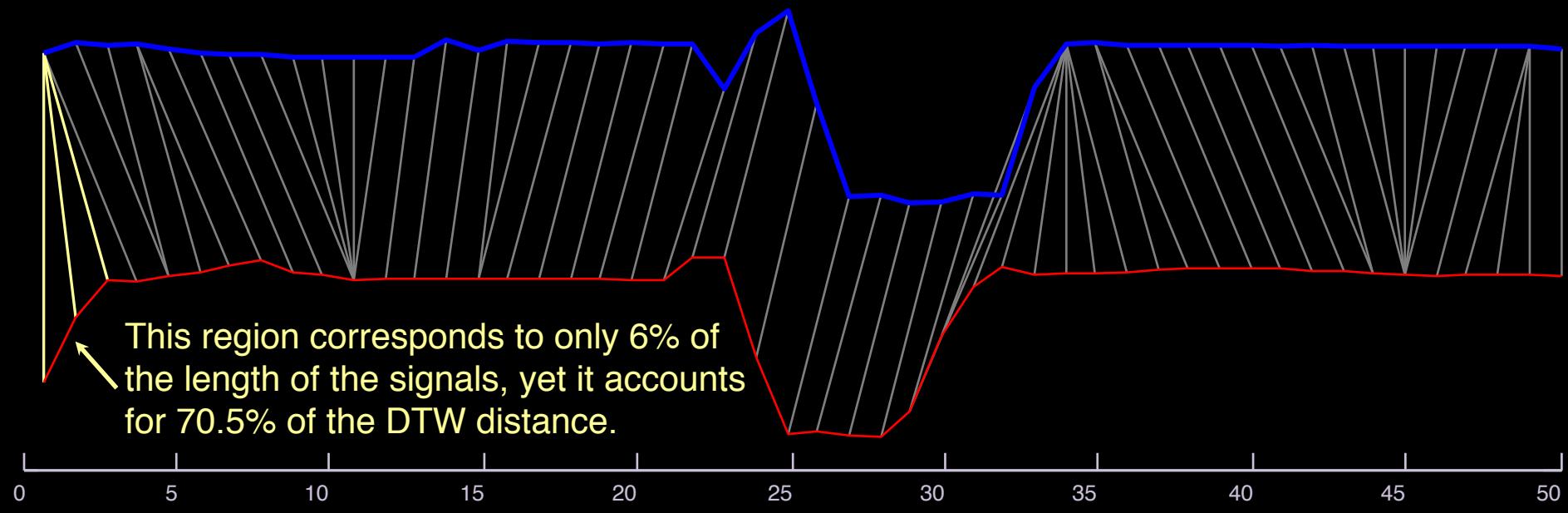
Thanks to Diego Silva for help with this example

The Importance of Endpoints: 2 of 3

DTW *can* be invariant to all the warping, but it cannot handle the difference at the very beginning of these two time series. Recall the boundary constraint:

Boundary Condition: $w_1 = (1,1)$ and $w_K = (m,n)$, this requires the warping path to start and finish in diagonally opposite corner cells of the matrix.

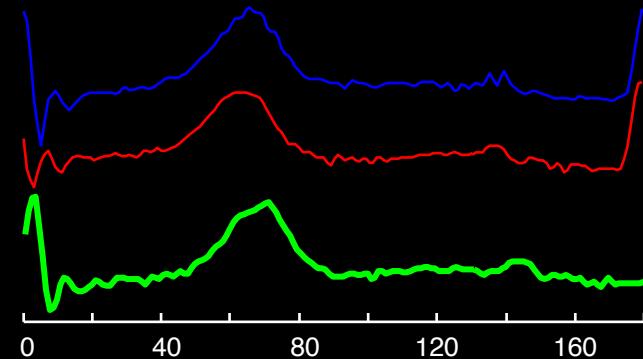
This constraint means that DTW must match the pairs of *beginning* (and *end*) points, even though they may be a poor match, as here.



The Importance of Endpoints: 3 of 3

Possible Fixes (Open problem)

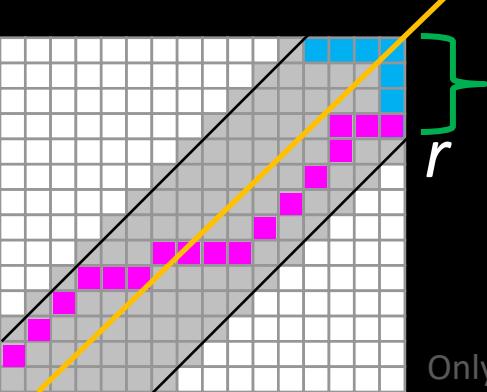
1) Use a better subsequence extraction algorithm.
For example, the heartbeats to the right were very badly extracted, and have a large variability in the endpoints.



2) Recall the warping path is:

$$W = w_1, w_2, w_3, \dots, w_{K-2}, w_{K-1}, \dots, w_K$$

If you fear that the error is concentrated at the ends, you can multiple the first and last r elements by a weight \hat{W} , with $0 \leq \hat{W} \leq 1$



3) Open-Ended Warping: Redefine the endpoint constraint. Change...

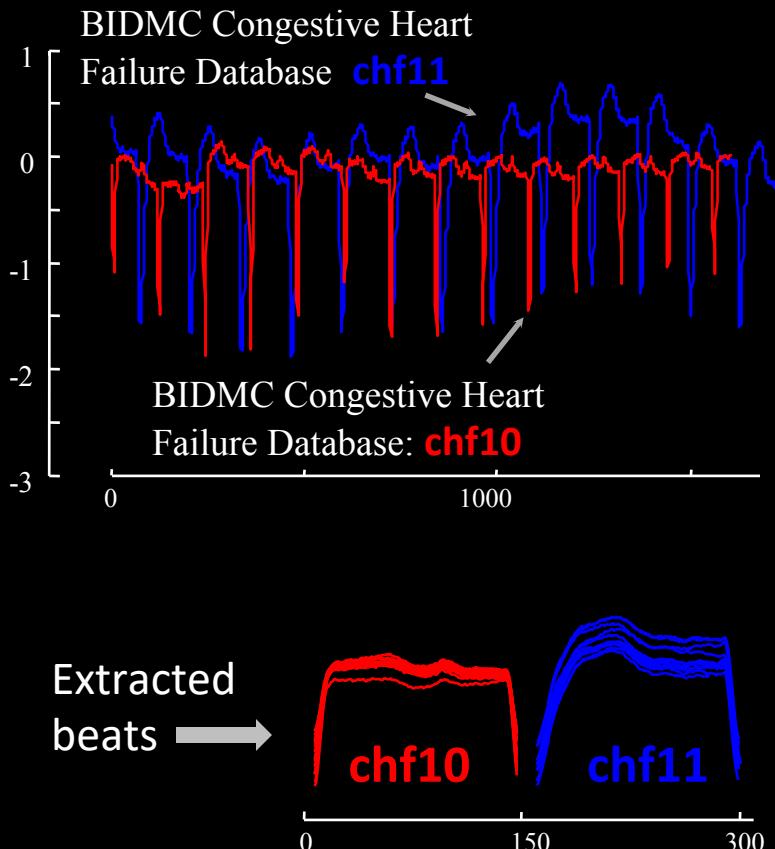
Boundary conditions: $w_1 = (1,1)$ and $w_K = (m,n)$, this requires the warping path to start and finish in diagonally opposite corner cells of the matrix.

...to

Boundary conditions: $w_1 = (1,B)$ or $(B,1)$ and $w_K = (m-C,n)$ or $(m,n-C)$, with $0 \leq B \leq r$, and $0 \leq C \leq r$ this requires the warping path to start and finish in a cyan cell.

Only top right section of the warping matrix shown for brevity

The Importance of Z-Normalization 1 of 3



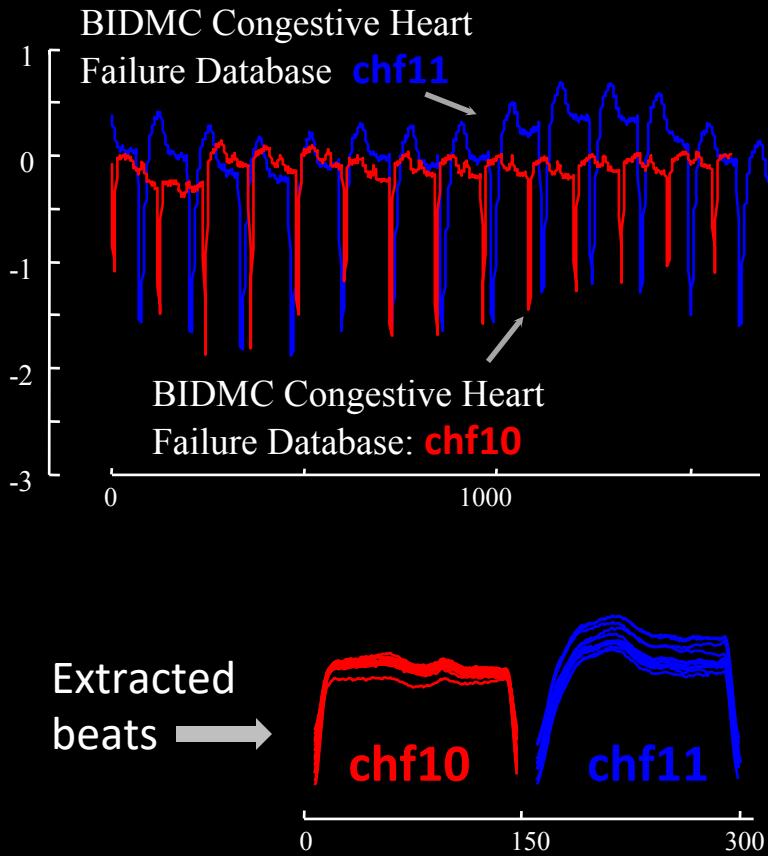
Essentially all datasets must have *every* subsequence z-normalized.

There are a handful of occasions where it does not make sense to z-normalize, but in those cases, DTW probably does not make sense either.

In this example, we begin by extracting heartbeats from two unrelated people.

Even without normalization, it happens that both sets have almost the same mean and standard deviation. Given that, do we need to bother to normalize them? (next slide)

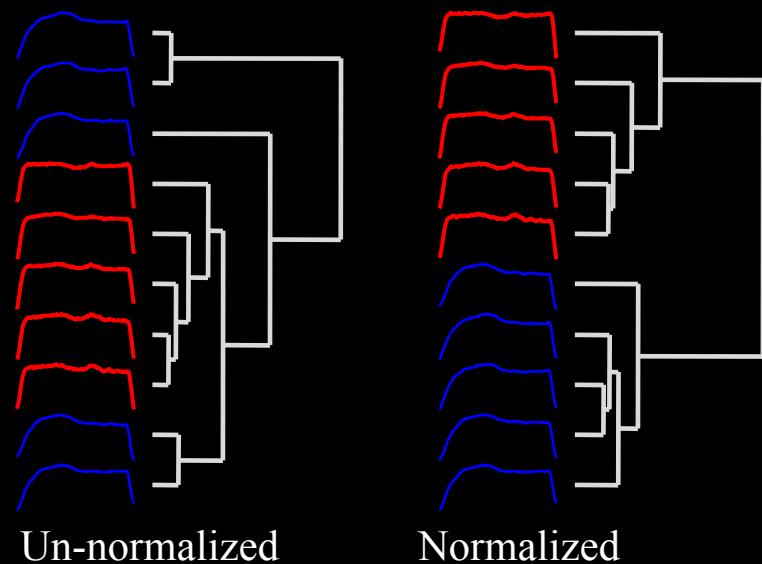
The Importance of Z-Normalization 2 of 3



Surprisingly z-normalizing can be a computational bottleneck, but later we will show you how to fix that.

Without normalization, the results are *very poor*, some blue heartbeats are closer to red heartbeats than there are to another blue beat .

With normalization, the results are perfect.

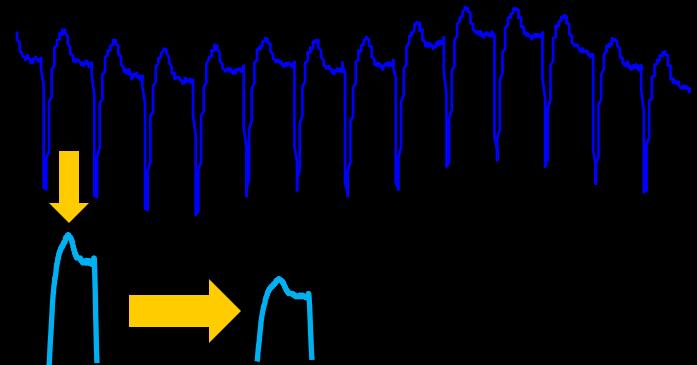


In this example, we extracted heartbeats from two different time series, and clustered them with and without normalization.

The Importance of Z-Normalization 3 of 3

Preempting a common misunderstanding

It is *not* sufficient to normalize the entire time series. You must normalize each subsequence



While there are a handful of normalization techniques , in most cases z-normalizing seems best:

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

Consider for example the famous Gun/Point dataset. Do we need to normalize it?

Not as it stands... ...but suppose

- The camera pans down? (the mean changes)
- The camera zooms out? (the STD changes)
- The actor wears high heel shoes? (the mean changes)
- We find a shorter actor? (the mean and the STD change)



Can you beat 1NN-DTW?

Lots of papers claim to be able to beat 1NN-DTW, is that true?

Hmm..., no and yes.

- **No:** More than 90% of such claims are false to cherry-picking, training and testing on the same data, the Texas Sharpshooter fallacy (see next slide) , etc
- **Yes:** There are a *handful* of such methods, but...
 - They *sometimes* win, but typically by very slim margins (say 1 to 3% better).
 - They come at a huge cost of coding effort, time complexity.
 - They *just* beat the most basic 1NN-DTW, with the simple CV trick to learn the warping window. It is less clear they could beat KNN-DTW, when a little more effort was spent to find a good warping window.
 - We strongly recommend the bake-offs/discussion by Dr. Tony Bagnall.

At a minimum, it is clear that 1NN-DTW will get you within 98% of the best accuracy possible, in the first five minutes.

Can you beat 1NN-DTW?

The Texas Sharpshooter Fallacy

- A paper in SIGMOD 2016 claims “*Our STS3 approach is more accurate than DTW in our suitable scenarios*”.
- They then note “*DTW outperforms STS3 in 79.5% cases.*” !?!
- They then do a ***post-hoc*** explanation of why they *think* they won on 20.5% of the cases that “suit them”.
- The problem is the ***post-hoc*** analysis, this is a form of the Texas Sharpshooter Fallacy. Below is a visual representation.

This is what they show you,
and you are impressed...



Can you beat 1NN-DTW?

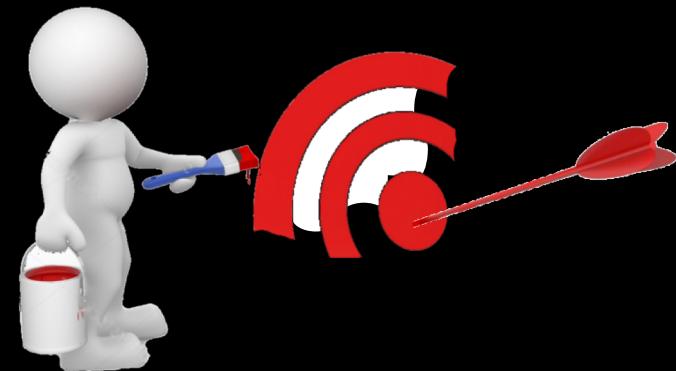
The Texas Sharpshooter Fallacy

- A paper in SIGMOD 2016 claims “*Our STS3 approach is more accurate than DTW in our suitable scenarios*”.
- They then note “*DTW outperforms STS3 in 79.5% cases.*” !?!
- They then do a ***post-hoc*** explanation of why they *think* they won on 20.5% of the cases that “suit them”.
- The problem is the ***post-hoc*** analysis, this is a form of the Texas Sharpshooter Fallacy. Below is a visual representation.

This is what they show you,
and you are impressed...



...until you realize that they shot the arrow first, and then painted the target around it!



Can you beat 1NN-DTW?

A good visual trick to compare algorithms of the 80 or so labeled time series in the public domain is the Texas Sharpshooter plot.

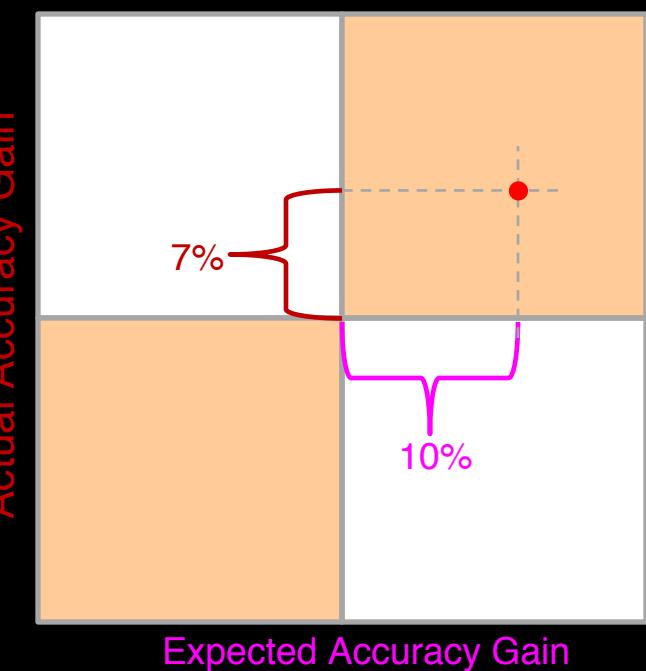
For each dataset

- First you compute the baseline accuracy of the approach you hope to beat.
- Then you compute the **expected improvement** we would get using your proposed approach (at this stage, learning any parameters and settings) using *only* the training data. Note that the expected improvement could be negative.
- Then compute the **actual improvement** obtained (using these now hardcoded parameters and settings) by testing on the test dataset.

You can plot the point {**expected improvement** , **actual improvement**} in a 2D grid, as below.

In this example, we predicted the **expected improvement** would be 10%, and the **actual improvement** obtained was 7%, pretty close!

We need to do these for all 80 or so datasets. What are the possible outcomes?



Can you beat 1NN-DTW? 3 of 3

With a Texas Sharpshooter plot, each dataset falls into one of four possibilities.

- We expected an improvement and we got it! This is clearly the best case.
- We expected to do worse, and we did. This is *still* a good case, we know not to use our proposed algorithm for these datasets
- We expected to do worse, but we did better. This is the wasted opportunity case.
- We expected to do better, but actually did worse. This is the worst case.

Now that we know how to read the plots, we will use it to see if DTW is better than Euclidean Distance,

Expected Improvement: We will search over different warping window constraints, from 0% to 100%, in 1% increments, looking for the warping window size that gives the highest 1NN training accuracy (if there are ties, we choose the smaller warping window size).

Actual Improvement: Using the warping window size we learned in the last phase, we test the holdout test data on the training set with 1NN.

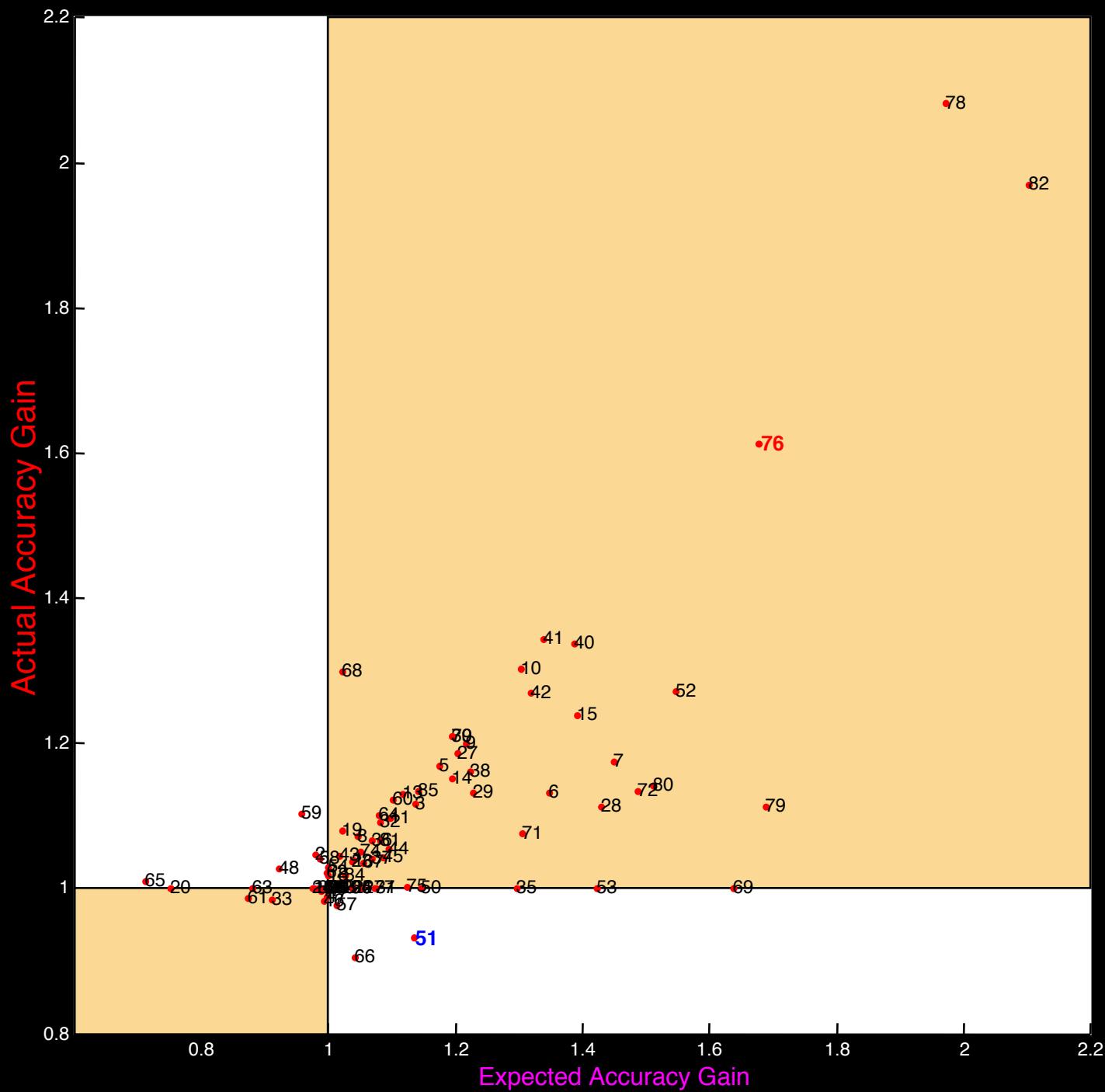


The results are strongly supportive of the claim “DTW better than Euclidean distance for most problems”

We sometimes had difficulty in predicting *when* DTW would be better/worse, but many of the training sets are tiny, making such tests very difficult.

For example, 51 is BeetleFy, with just 20 train and 20 test instances. Here we expected to do a little better, but we did a little worse.

In contrast, for 76 (LargeKitchenAppliances) we had 375 train and 375 test instances, and were able to more accurately predict a large improvement.



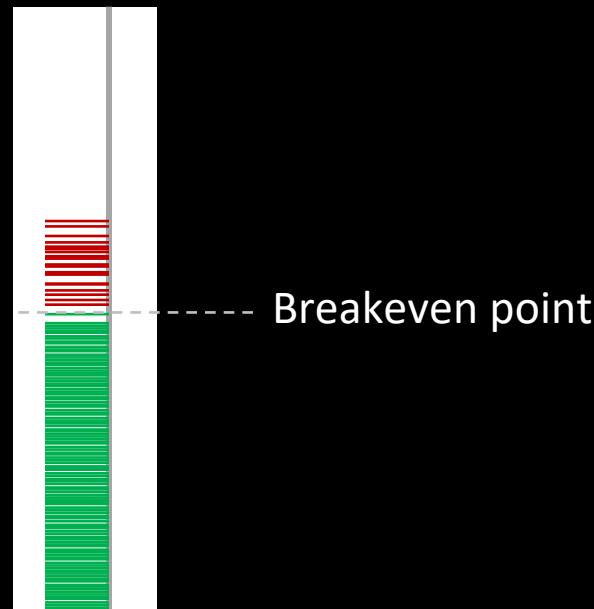
Can you beat 1NN-DTW?

Recall the paper in SIGMOD that claimed “*Our STS3 approach is more accurate than DTW in our suitable scenarios*”. And “*DTW outperforms STS3 in 79.5% cases.*”

They are claiming to be better for 1/5th of the datasets, but in essence they are only reporting one axis of the Sharpshooter plot.

They did (slightly) win 20.5% of the time.

They lost 79.5% of the time.



*We had considered trying to reimplement this work to make this plot. But the idea has 3 parameters, and the writing is quite vague..

Can you beat 1NN-DTW?

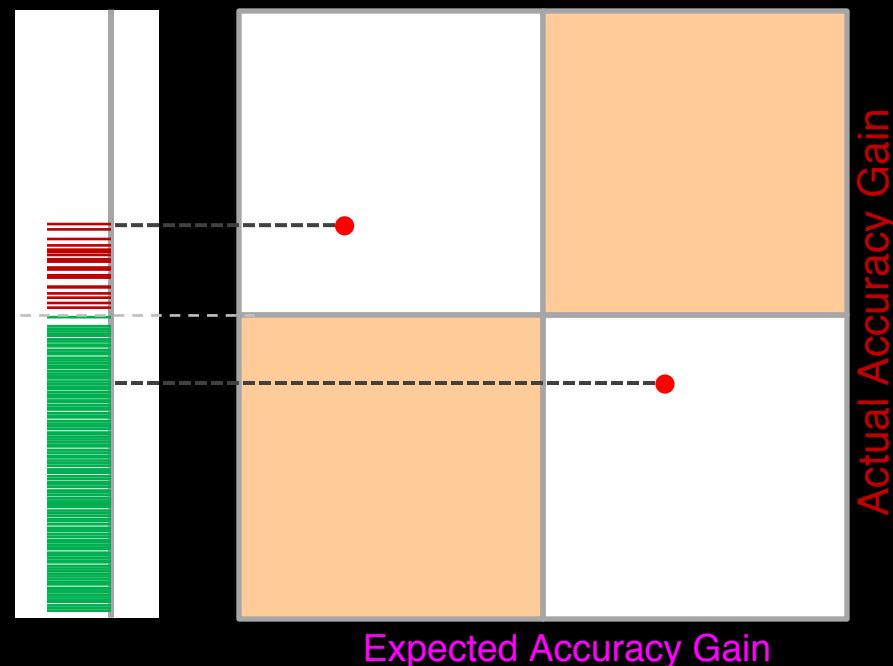
Recall the paper in SIGMOD that claimed “*Our STS3 approach is more accurate than DTW in our suitable scenarios*”. And “*DTW outperforms STS3 in 79.5% cases.*”

They are claiming to be better for 1/5th of the datasets, but in essence they are only reporting one axis of the Sharpshooter plot.

It may be * that if we computed the Sharpshooter plot it would look like the below (for two selected points only)

They did (slightly) win 20.5% of the time, but they did not predict ahead of time that they would win.

They lost 79.5% of the time.
Moreover, on a huge fraction of the datasets they lost on, they might have said “*you should use our algorithm here, we think we will win*”, and we would have been much worse off!



*We had considered trying to reimplement this work to make this plot. But the idea has 3 parameters, and the writing is quite vague..

Can you beat 1NN-DTW?

Summary for this section:

I think that at least 90% of the claims to beat DTW are wrong.

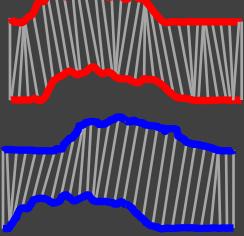
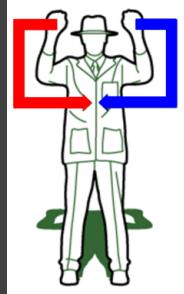
Of the 10% of claims that remain

- They are beating the simplest 1NN-DTW with w learned in a simple way. Using KNN-DTW , smoothing the data, relaxing the endpoint constraints, better methods for learning w etc, would often close some or all the gap.
- The improvements are so small in most cases, it takes a sophisticated and sensitive test to be sure you have a real improvement.

The Texas Sharpshooter test is a great sanity check for this, but you should see the work of Anthony Bagnall and students

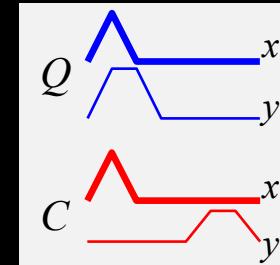
<https://arxiv.org/abs/1602.01711> for a more principled methods.

Generalizing to Multi-Dimensional Data 1 of 4



It is increasingly common to encounter Multi-Dimensional (MD) time series data. Here we measure the X-axis acceleration of both the **left** and **right** hand.

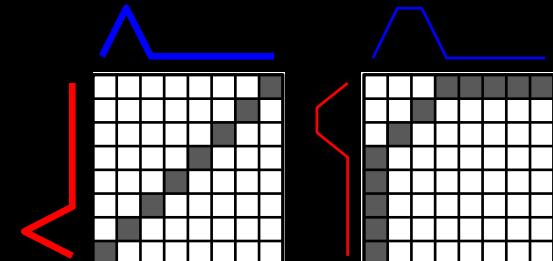
Given these pair of 2D objects...



There are two obvious ways to compute the MD DTW score.

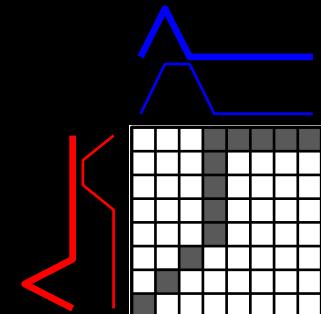
Independent: Just compute the DTW score for each dimension independently, and sum up each score.

$$\text{DTW}_I(Q, C) = \text{DTW}(Q_x, C_x) + \text{DTW}(Q_y, C_y) = 2.4$$



Dependent: Create a single distance matrix that reflect the distance between each corresponding pair of time series, then find the *single* warping path and distance as per normal.

$$\text{DTW}_D(Q, C) = \text{DTW}(\{Q_x, Q_y\}, \{C_x, C_y\}) = 3.2$$



Generalizing to Multi-Dimensional Data 2 of 4

So, of DTW_I and DTW_D which is best?

Lets think of it this way:

The thing we want classify is an physical process, the utterance of the word “*bicycle*”, the beat of a heart, an autograph, a tennis shot etc.

We cannot see the actual event, just 2 or more time series it created.

- If the physical process affects the time series simultaneously, then DTW_D will probably be best. We call this the *tightly coupled* case.
- If the physical process affects the time series with varying lags, then DTW_I will probably be best. We call this the *loosely coupled* case.

Example: Suppose we measure the directionless acceleration of the left and right wrists of a tennis player.

The “physical process” is a *backhand stoke*. If a two-handed backstroke, the two time series are tightly coupled. If a one-handed backhand, the two hands will be very loosely coupled .

Jo-Wilfried Tsonga



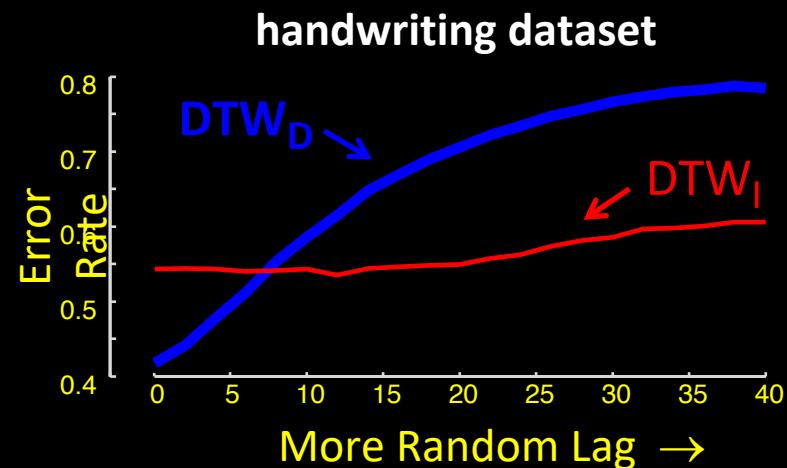
Generalizing to Multi-Dimensional Data 3 of 4

We can demonstrate the claim in the last slide with experiments.

Let us begin with a dataset that we are 100% sure is tightly coupled, then slowly add some random time lags into the data...

We know that the **handwriting dataset** is *tightly coupled*. We find that DTW_D has an error-rate of about 0.42, much better than that DTW_I has an error-rate of about 0.55.

However, as we uncouple the perfect synchronization by adding some random lag, DTW_D quickly gets worse, while that DTW_I is barely effected.



Word Recognition from Articulatory Movement Data

Data	$\text{DTW}(1\text{st})$	$\text{DTW}(2\text{nd})$	DTW_I	DTW_D
$T1_Z \text{ UL}_X$	0.34	0.59	0.25	0.31
$T1_Y \text{ T1}_Z$	0.38	0.34	0.24	0.15

A diagram of a human head in profile, facing right. Points are labeled on the head and mouth area: HR (head rotation), HC (head center), UL (upper lip), LL (lower lip), T1 (teeth), I (inner teeth), F (front teeth), and H (high teeth). A small coordinate system (y, z, x) is shown to the left of the head.

This effect is observed on real data

Note the results here show that..

- 1) Using 2D *can* be better than 1D
- 2) Neither DTW nor DTW_I dominates

Generalizing to Multi-Dimensional Data 4 of 4

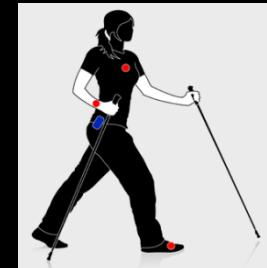
Critical Point: You *can* generalize DTW to 2,3,4,...1,000 dimensions.

However, it is *very* unlikely that more than 2 to 4 is useful, after that, you are almost certainly condemned to the curse of dimensionality.

Consider a physical activity dataset containing 36 axis synchronous measurements from three Inertial Measurement Units (IMUs) located on the wrist, chest and ankle. This dataset has eight subjects performing activities such as: rope-jumping, running, folding laundry, ascending-stairs.

Using *all* dimensions is a disaster!
Using the best three is a lot better than using the best one, so there is evidence that Multi-Dimensional DTW really does help.

PAMAP, Physical Activity Monitoring for Aging People www.pamap.org/
(we used DTW with $w = 0$ for simplicity here, we could do better by tuning w)



Which Dimensions?	Accuracy
All dimensions	0.19
A single random dimension (on average)	0.51
The best single dimension (as predicted by CV)	0.72
The best 3 dimensions (as predicted by CV)	0.89

Summary...



Flat-tailed Horned Lizard

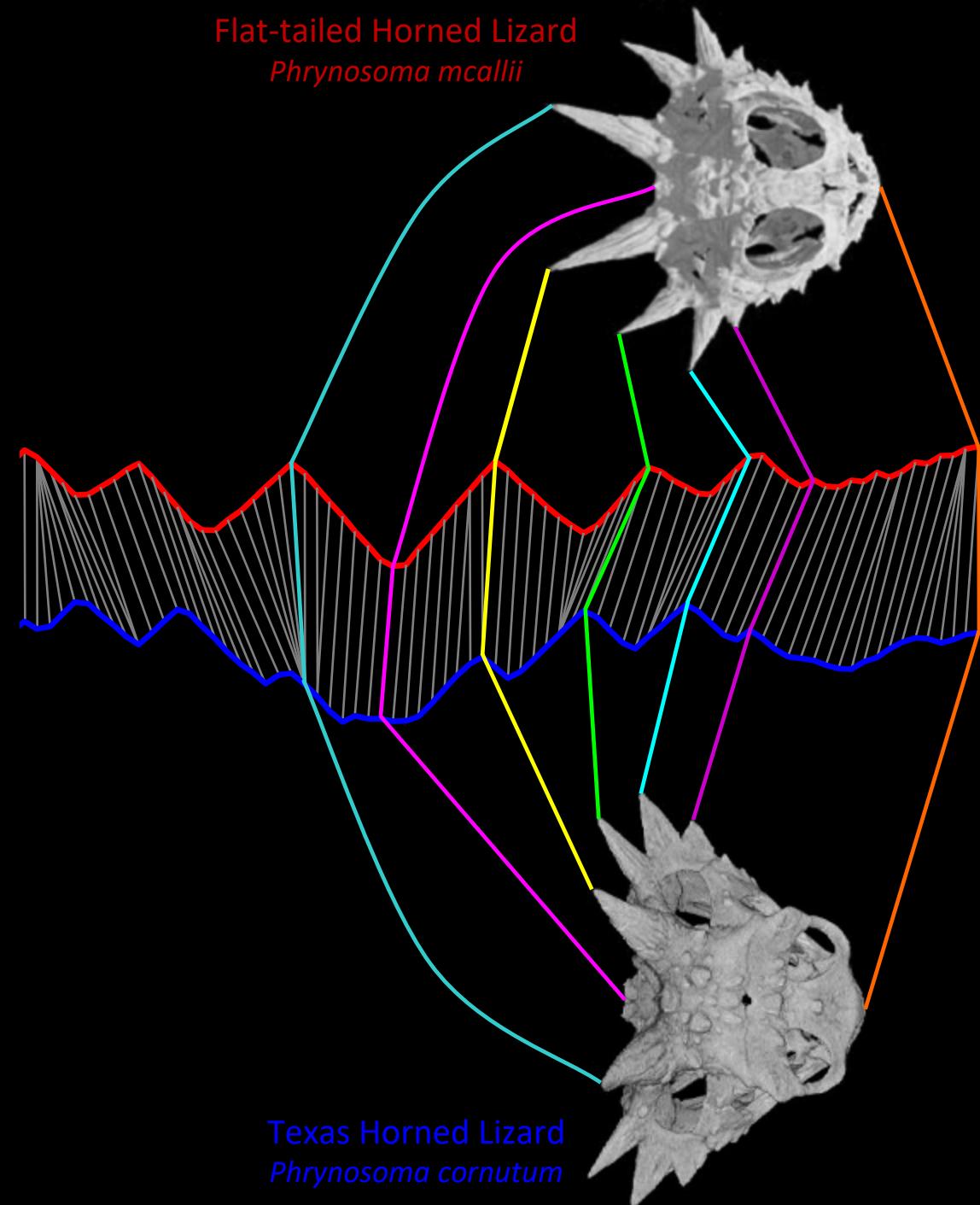
Phrynosoma mcallii

Summary...

DTW is an extraordinarily powerful and useful tool.

Its uses are limited only by our imaginations.

We believe it will remain an important part of the data mining toolbox for decades to come.



Coffee Break!

When we come back after the break

- Dr. Mueen will take over
- We will learn how to speed up DTW, and scale up to a trillion subsequences!



The Second Act: How to do DTW *fast*

- We are motivated that DTW is GOOD by the first act
- The general conception: *DTW is slow and we have a never-ending need for speed*
 - Better performance in knowledge extraction
 - Better scalability to process BigData
 - Better interactivity in human driven data analysis

What can be made fast?

- One-to-One comparison
 - Exact Implementation and Constraints
 - Efficient Approximation
 - Exploiting Sparsity
- One-to-Many comparisons
 - Nearest Neighbor Search
 - In a database of independent time series
 - In subsequences of a long time series
 - Density Estimation
 - In clustering
 - Averaging Under Warping
 - In classification
- Many-to-Many comparisons
 - All-pair Distance Calculations

Speeding up DTW: one-to-one

- One-to-One comparison
 - Exact Implementation
 - Efficient Constraints
 - Exploiting Hardware
 - Efficient Approximation
 - Exploiting Sparsity

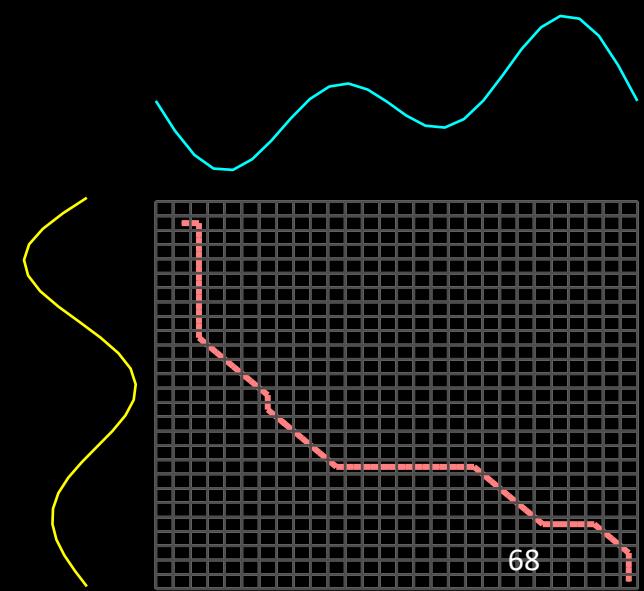
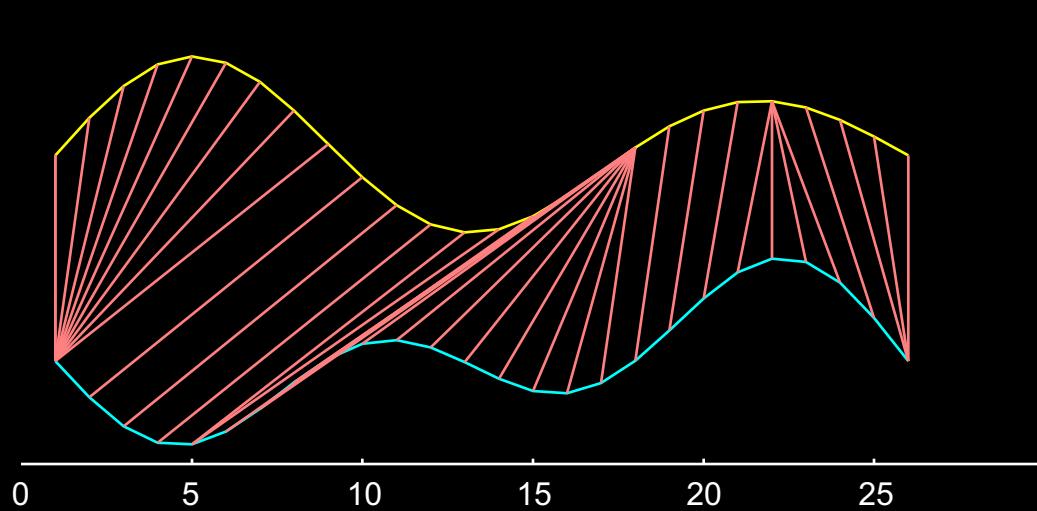
Simplest Exact Implementation

Input: x and y are time series of length n and m

Output: DTW distance d between x and y

```
D(1:n+1,1:m+1) = inf;  
D(1,1) = 0;  
for i = 2 : n+1 %for each row  
    for j = 2 : m+1 %for each column  
        cost = (x(i-1)-y(j-1))^2;  
        D(i,j) = cost + min( [ D(i-1,j), D(i,j-1), D(i-1,j-1) ] );  
d = sqrt(D(n+1,m+1));
```

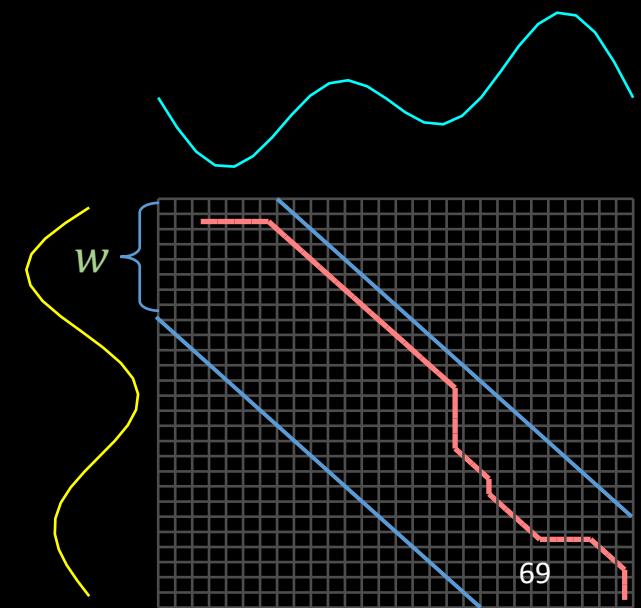
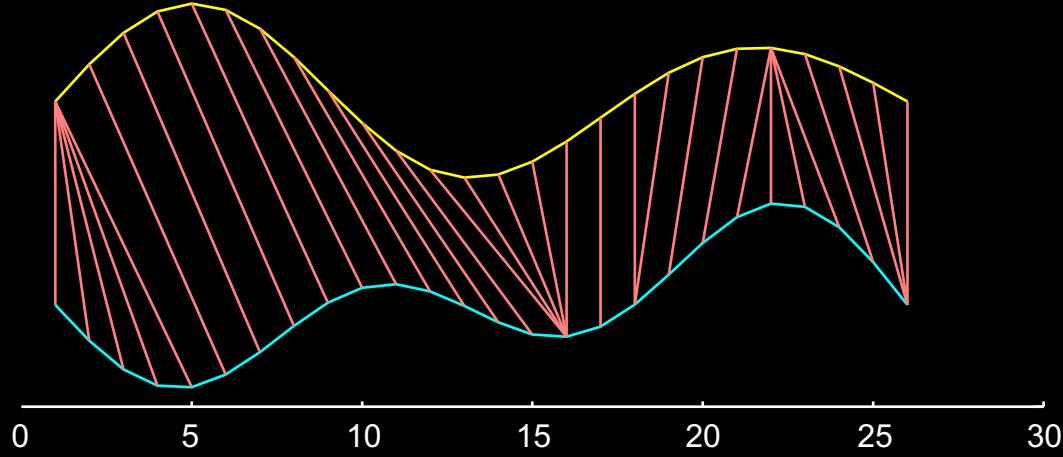
$O(n^2)$ time
 $O(n^2)$ space



Simplest Implementation (Constrained)

```
D(1:n+1,1:m+1) = inf;  
D(1,1) = 0;  
w = max(w, abs(n-m));  
for i = 2 : n+1  
    for j = max(2,i-w) : min(m+1,i+w)  
        cost = (x(i-1)-y(j-1))^2;  
        D(i,j) = cost + min( [ D(i-1,j), D(i,j-1), D(i-1,j-1) ] );  
d = sqrt(D(n+1,m+1)); 6969
```

$O(nw)$ time
 $O(n^2)$ space



Memoization

```
D(2,1:m+1) = inf;  
D(2,1) = inf;  
D(1,1) = 0;  
p = 1; c = 2;  
for i = 2 : n+1  
    for j = 2 : m+1  
        cost = (x(i-1)-y(j-1))^2;  
        D(c,j) =  cost + min( [ D(p,j), D(c,j-1), D(p,j-1) ] );  
        swap(c,p);  
d = sqrt(D(n+1,m+1));
```

$O(n^2)$ time
 $O(n)$ space

Previous Row



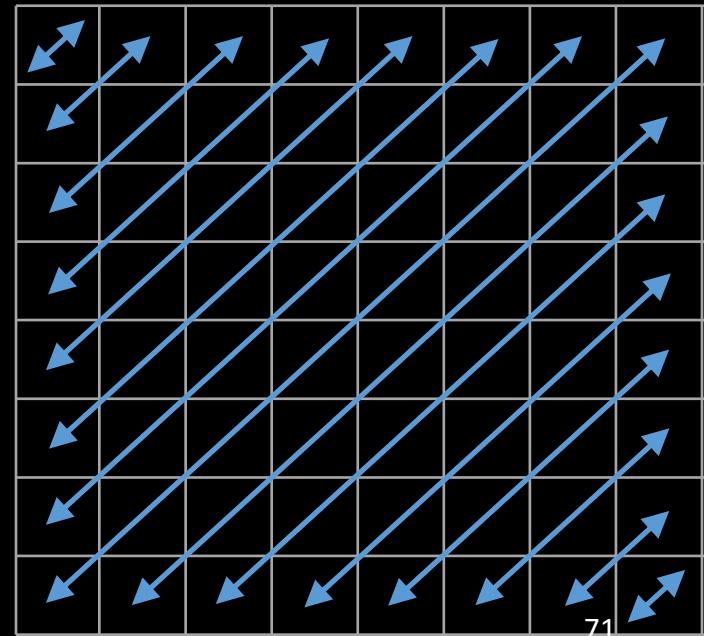
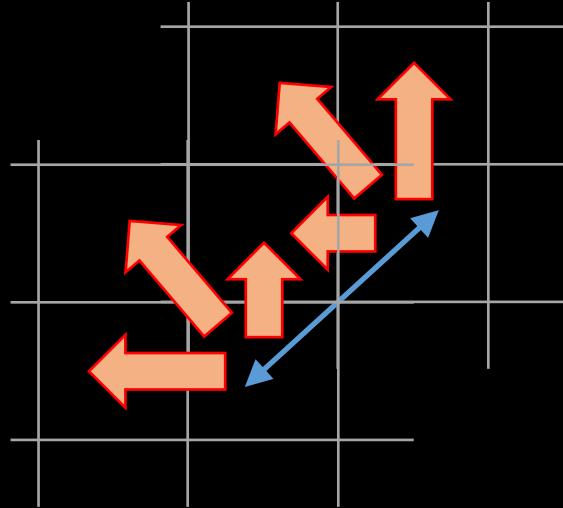
∞	1	2	2	1	2	3	4	5	3	2	1
∞	1	3	3								

Current Row

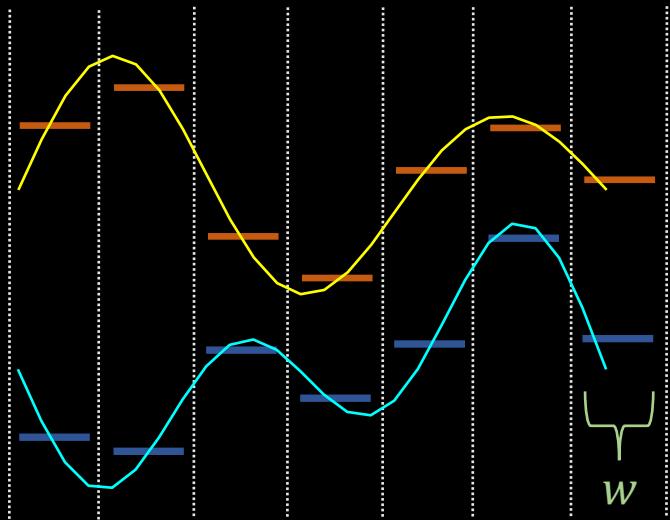
Hardware Acceleration

- Single Instruction Multiple Data (SIMD) architecture
- Cells on a diagonal are computed in parallel
- Values of a diagonal depend on the previous two diagonals

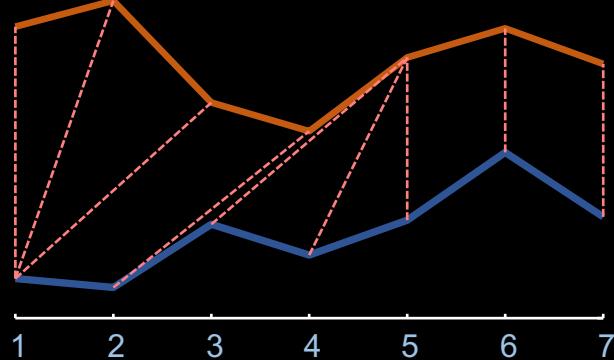
$O(n)$ time
 $O(n)$ space



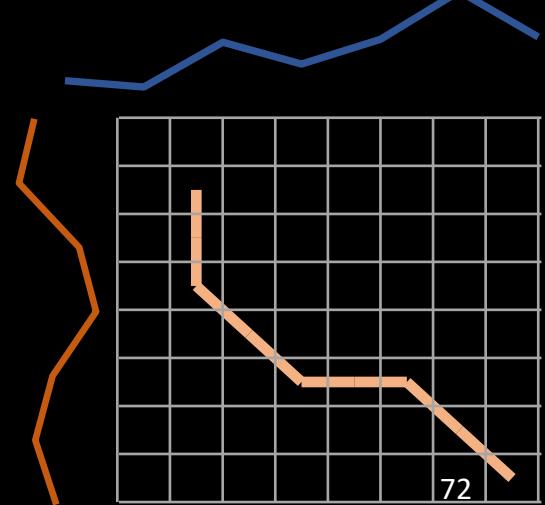
PAA based approximation



Piecewise Aggregate Approximation

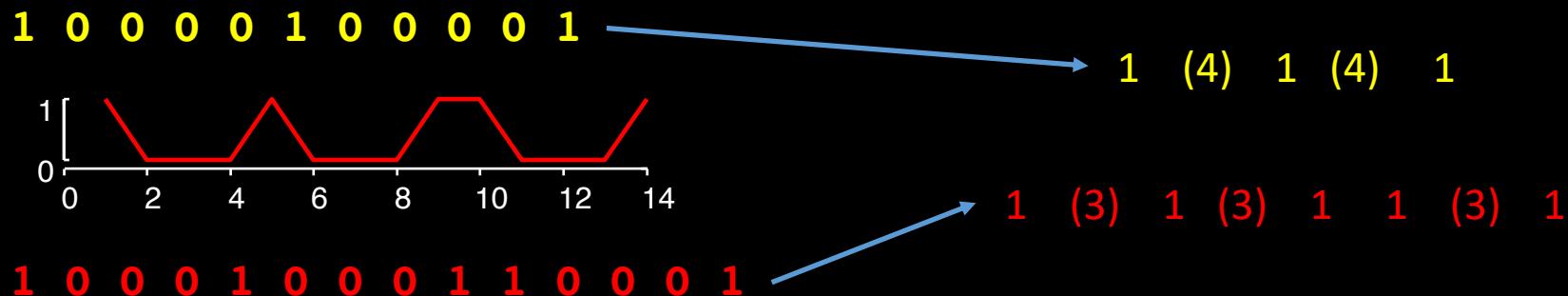
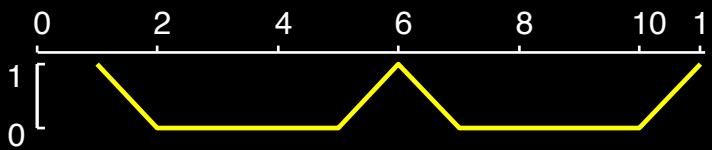


$O\left(\frac{n}{w}\right)^2$ time
 $O\left(\frac{n}{w}\right)^2$ space



Approximation by Length-encoding

To exploit sparsity,
encode lengths of the
runs of zeros

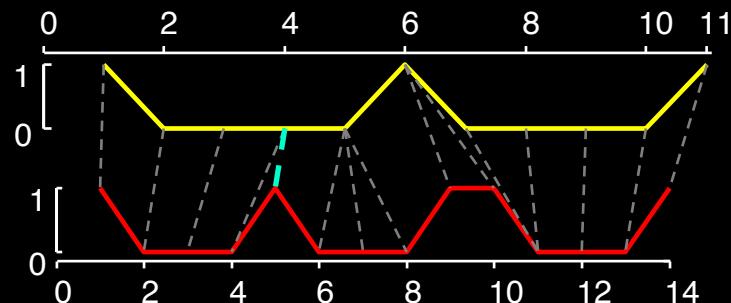


1 (4) 1 (4) 1

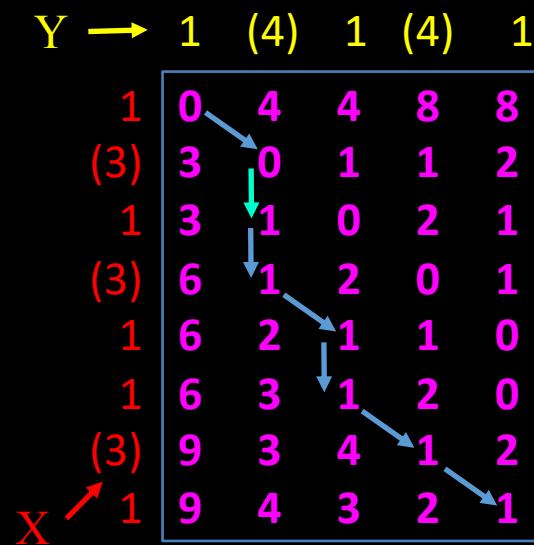
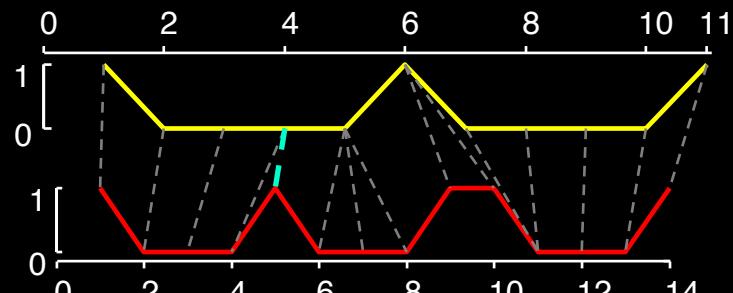
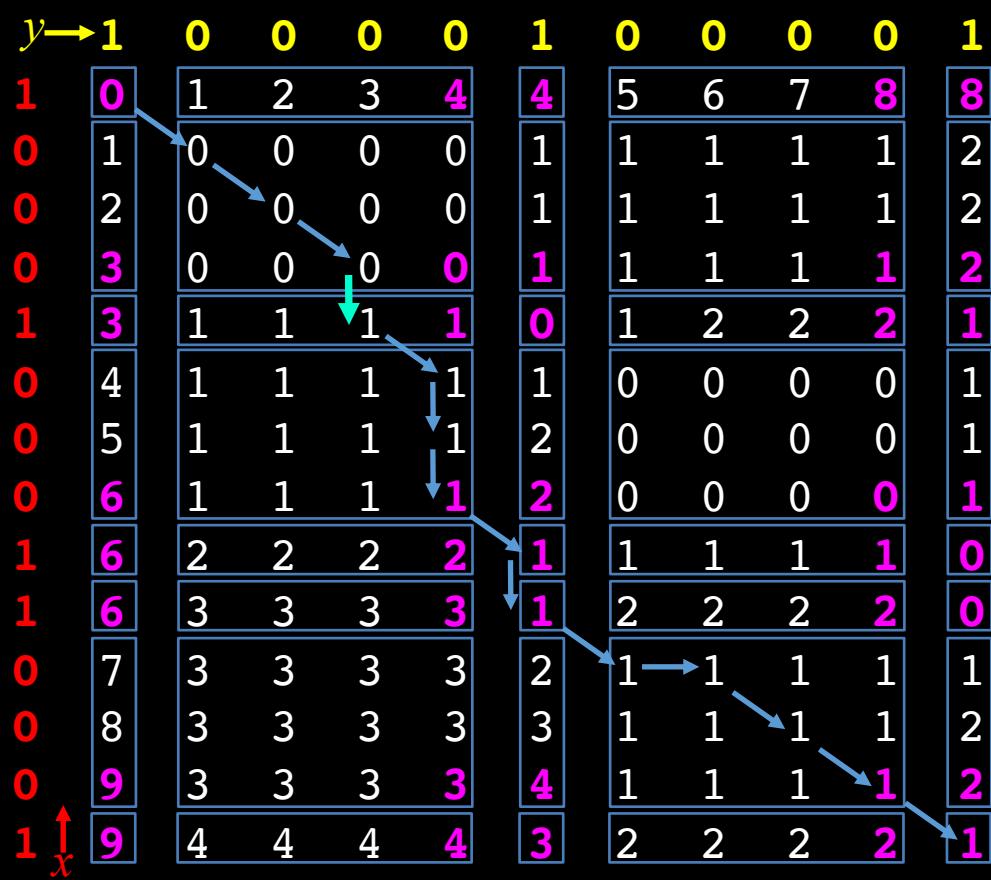
1 (3) 1 (3) 1 1 (3) 1

Exploiting Sparsity (1)

$y \rightarrow 1$	0	0	0	0	1	0	0	0	1
1	0	1	2	3	4	4	5	6	7
0	1	0	0	0	1	1	1	1	1
0	2	0	0	0	1	1	1	1	2
0	3	0	0	0	0	1	1	1	2
1	3	1	1	1	1	0	1	2	1
0	4	1	1	1	1	1	0	0	1
0	5	1	1	1	1	2	0	0	1
0	6	1	1	1	1	2	0	0	0
1	6	2	2	2	2	1	1	1	0
1	6	3	3	3	3	1	2	2	0
0	7	3	3	3	3	2	1	1	1
0	8	3	3	3	3	1	1	1	2
0	9	3	3	3	3	4	1	1	2
1	9	4	4	4	4	3	2	2	1

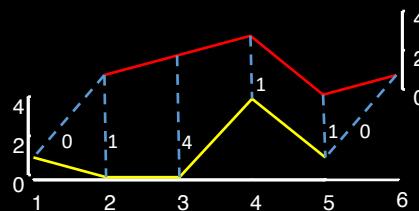


Exploiting Sparsity (2)

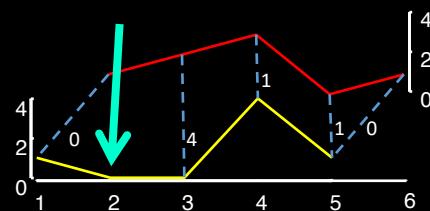


Exploiting Sparsity (2)

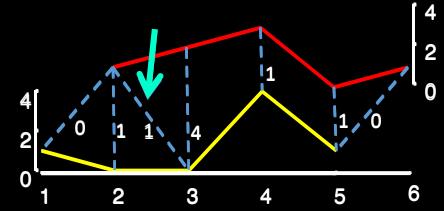
Correct Alignment



Missing Alignment



Extra Alignment



Non-Linear change
→ Exact Distance

$y \rightarrow$	1	0	0	4	1
$x \rightarrow$	0	1	2	11	11
1	4	5	6	7	
2	10	13	6	10	
3	5	5	21	7	
0	6	6	14		
1	6	6			

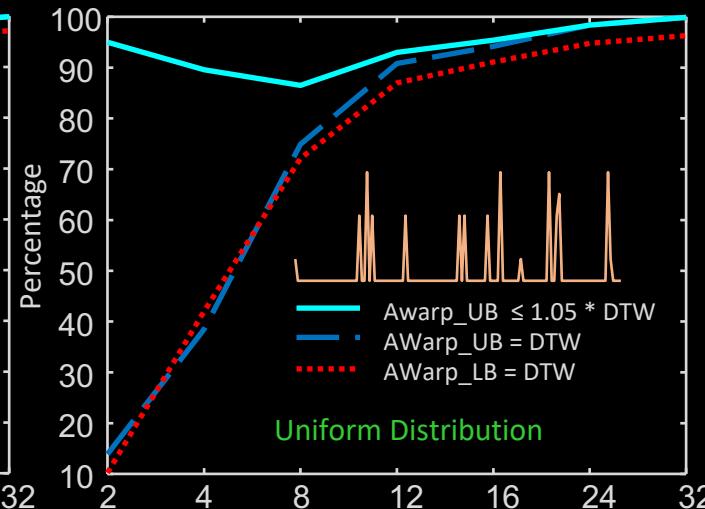
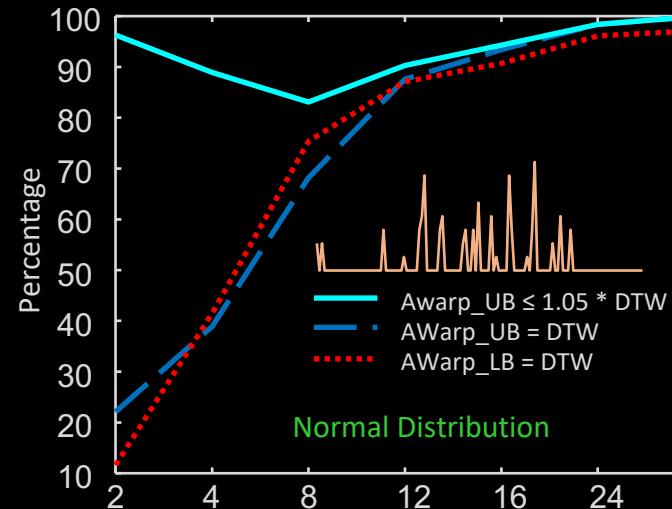
No change
→ Lower bound

$y \rightarrow$	1	(2)	4	1
$x \rightarrow$	0	2	11	11
1	4	6	7	
2	10	5	9	
3	5	21	6	
(1)	6	14		
1	6			

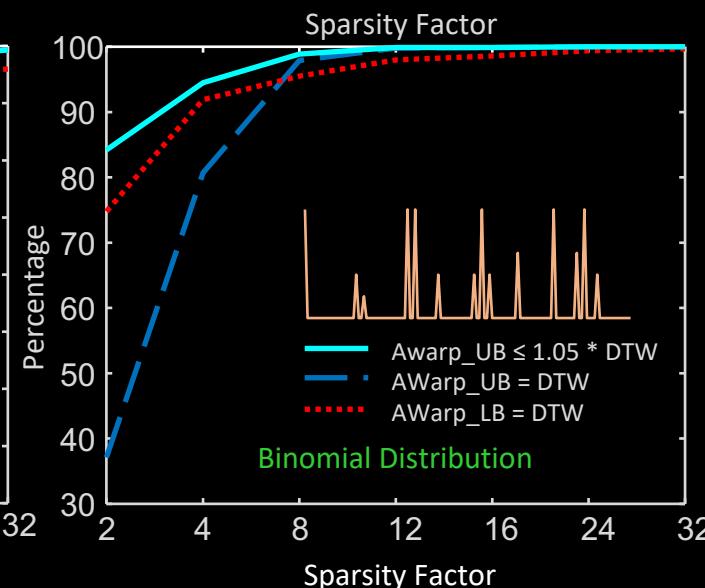
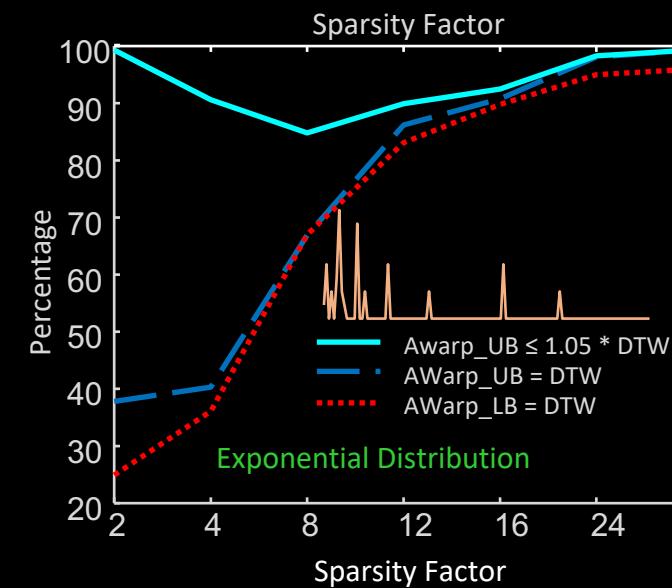
Linear change
→ Upper bound

$y \rightarrow$	1	(2)	4	1
$x \rightarrow$	0	2	11	11
1	6	6	7	
2	15	7	10	
3	5	21	8	
(1)	5	14	8	
1	6			

Exploiting Sparsity (3)



Sparsity Factor of s
means $\frac{1}{s}\%$ of the time
series is filled with non-
zeros.



What can be made fast?

- One-to-One comparison
 - Exact Implementation and Constraints
 - Efficient Approximation
 - Exploiting Sparsity
- One-to-Many comparisons
 - Nearest Neighbor Search
 - In a database of independent time series
 - In subsequences of a long time series
 - Density Estimation
 - In clustering
 - Averaging Under Warping
 - In classification
- Many-to-Many comparisons
 - All-pair Distance Calculations

Nearest Neighbor Search

- A query Q is given
- n independent candidate time series C_1, C_2, \dots, C_n
- $O(n)$ distance calculations are performed to

Find THE nearest neighbor of the
given query under DTW.

Brute Force Nearest Neighbor Search

Computational cost: $O(nm^2)$

Algorithm Sequential_Scan(Q)

```
1. best_so_far = infinity;  
2. for all sequences in database  
3.  
4.  
5.     true_dist = DTW( $C_i$ , Q);  
6.     if true_dist < best_so_far  
7.         best_so_far = true_dist;  
8.         index_of_best_match = i;  
9.     endif  
10.  
11. endfor
```

Lower Bounding Nearest Neighbor Search

We can speed up similarity search under DTW by using a lower bounding function

Algorithm Lower_Bounding_Sequential_Scan(Q)

```
1.   best_so_far = infinity;  
2.   for all sequences in database  
3.       LB_dist = lower_bound_distance( $C_i, Q$ );  
4.       if LB_dist < best_so_far  
5.           true_dist = DTW( $C_i, Q$ );  
6.           if true_dist < best_so_far  
7.               best_so_far = true_dist;  
8.               index_of_best_match = i;  
9.           endif  
10.      endif  
11.  endfor
```

Try to use a cheap lower bounding calculation as often as possible.



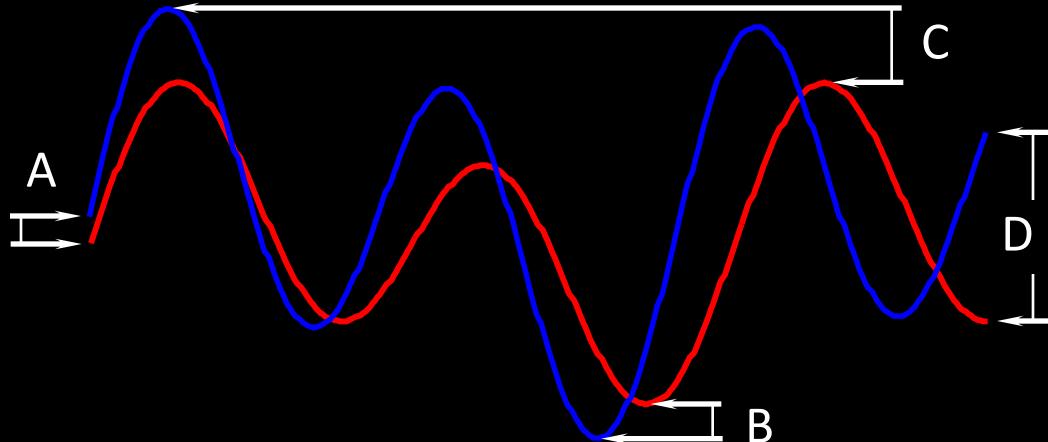
Only do the expensive, full calculations when it is absolutely necessary



Lower Bound of Kim



LB_Kim



$O(1)$ time if considered only first and last points

$O(n)$ time for all four distances

Kim, S, Park, S, & Chu, W. *An index-based approach for similarity search supporting time warping in large sequence databases*. ICDE 01, pp 607-614

The squared difference between the two sequence's first (A), last (D), minimum (B) and maximum points (C) is returned as the lower bound

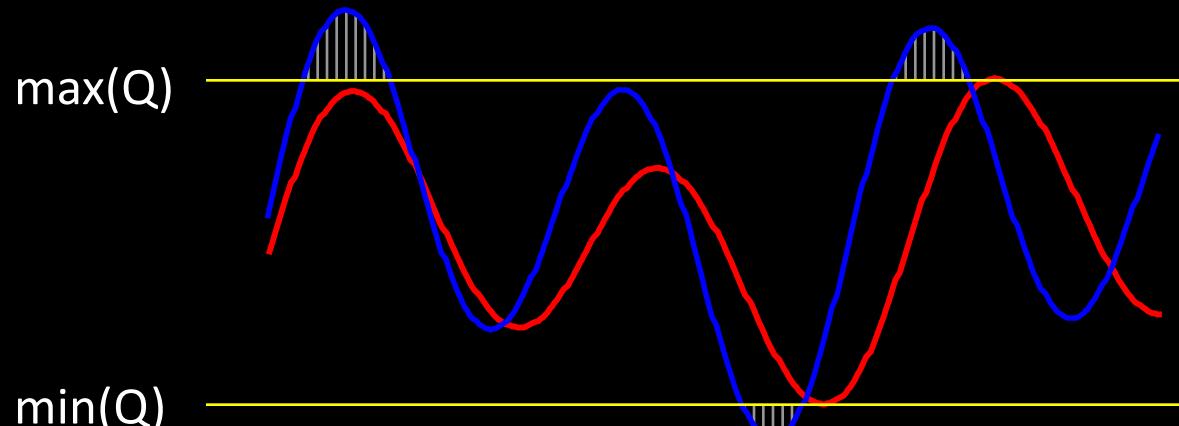
Lower Bound of Yi



LB_Yi

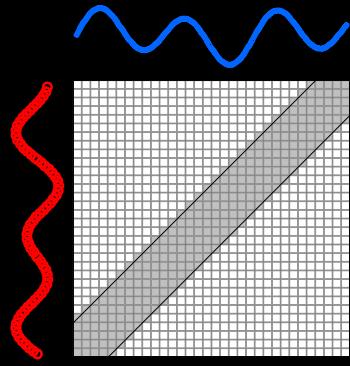
$O(n)$ time

Yi, B, Jagadish, H & Faloutsos, C.
Efficient retrieval of similar time sequences under time warping.
ICDE 98, pp 23-27.



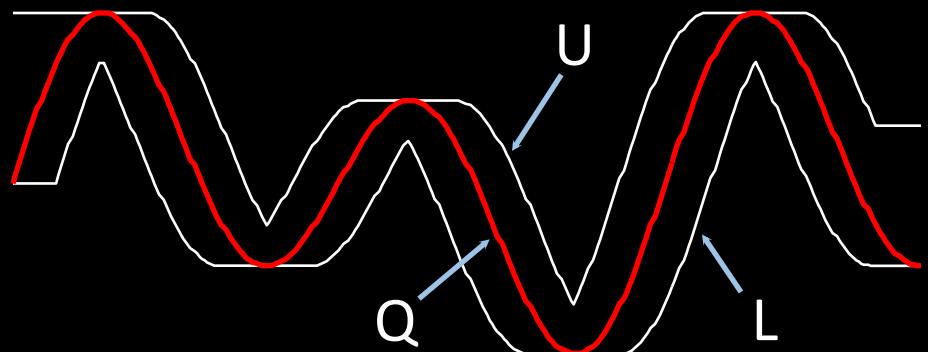
The sum of the squared length of white lines represent the minimum contribution of the observations above and below the **yellow** lines.

Lower Bound of Keogh



Sakoe-Chiba Band

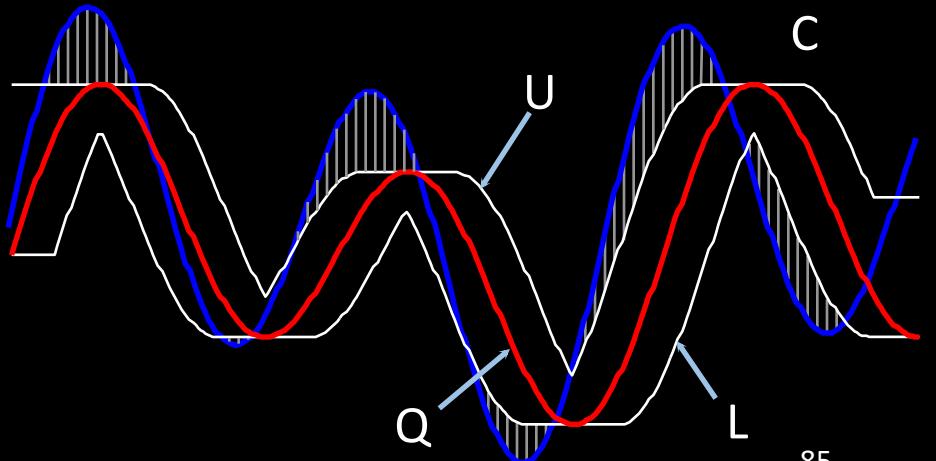
$$U_i = \max(q_{i-w} : q_{i+w})$$
$$L_i = \min(q_{i-w} : q_{i+w})$$



$O(n)$ time

Envelope-Based Lower Bound

$$LB_Keogh(Q, C) = \sum_{i=1}^n \begin{cases} (q_i - U_i)^2 & \text{if } q_i > U_i \\ (q_i - L_i)^2 & \text{if } q_i < L_i \\ 0 & \text{otherwise} \end{cases}$$

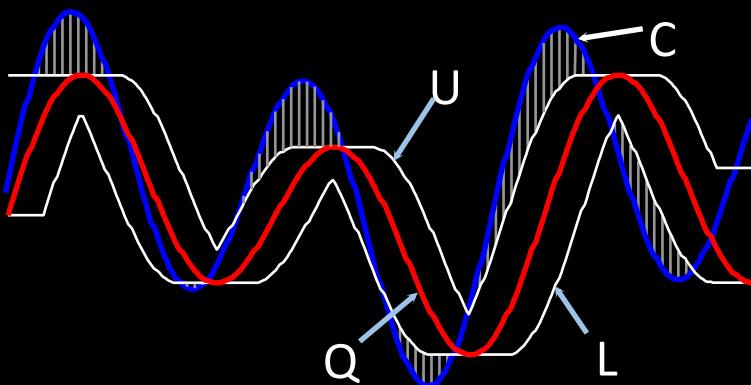


Reversing the Query/Data Role in LB_Keogh

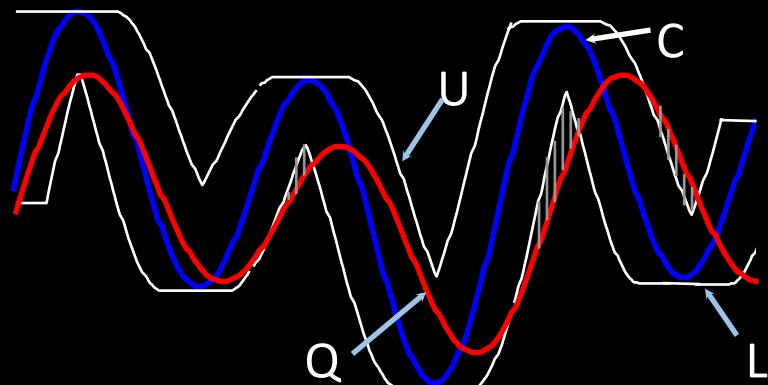
- Make LB_Keogh tighter
- Much cheaper than DTW
- U/L envelops on the candidates can be calculated online or pre-calculated

$$\max(\text{LB}_{\text{Keogh}}^{\text{EQ}}, \text{LB}_{\text{Keogh}}^{\text{EC}})$$

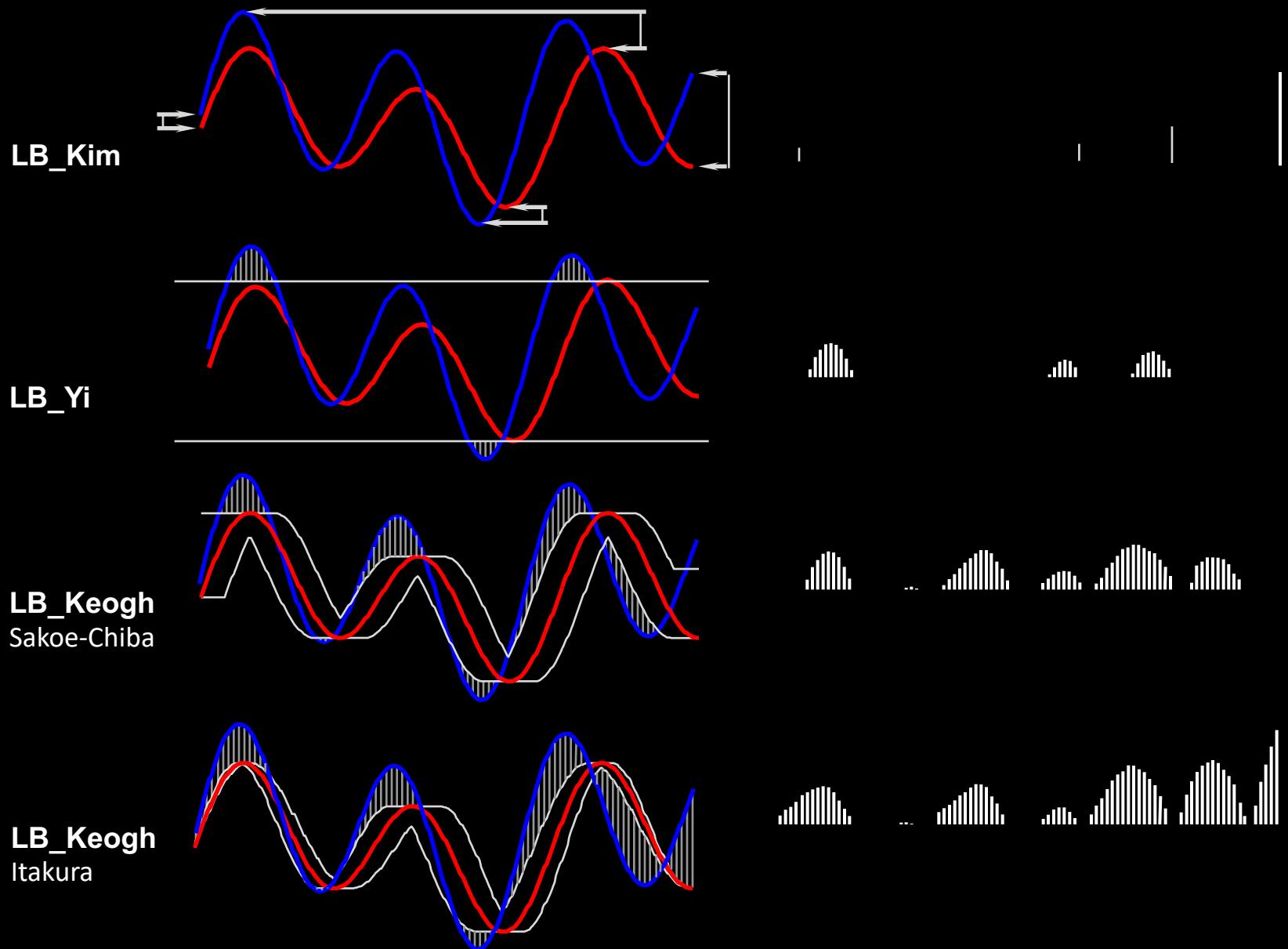
Envelop on Q



Envelop on C

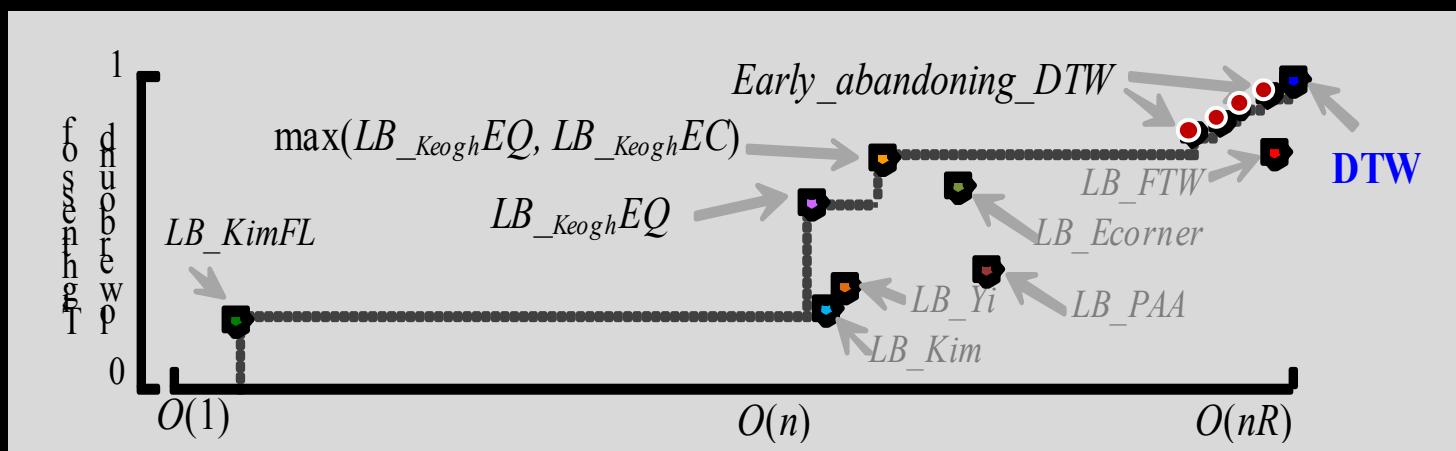


The tightness of the lower bound for each technique is proportional to the length of lines used in the illustrations



Cascading Lower Bounds

- At least 18 lower bounds of DTW was proposed.
- Use lower bounds only on the **Skyline**.
- Use the bounds on the skyline in cascade from least expensive to most expensive
- When unable to prune, use early abandoning techniques



99.9% of the time DTW is not calculated₈₈

Early Abandoning Techniques

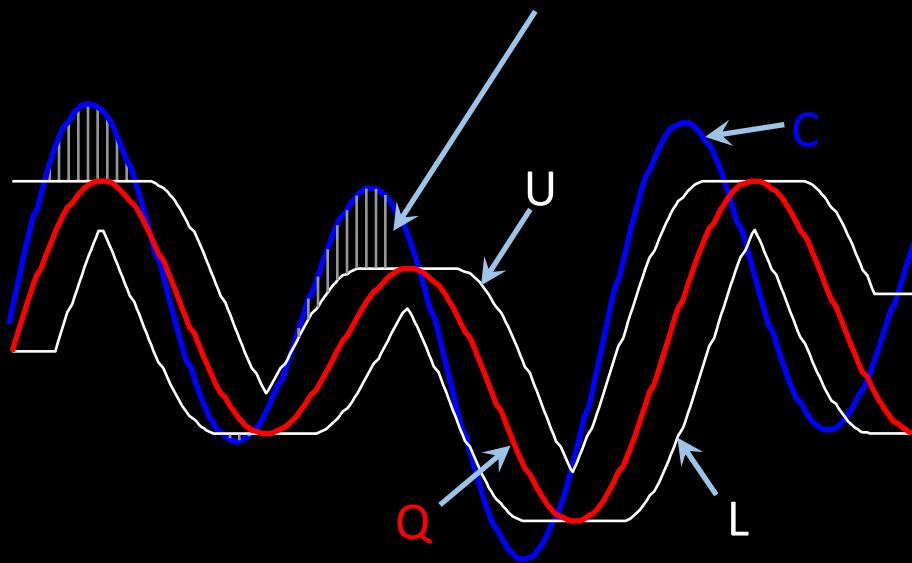
Abandon accumulating errors as soon as the current total is larger than the *best_so_far*

Four techniques to abandon early

1. Early Abandoning of LB_Keogh
2. Early Abandoning of DTW
3. Earlier Early Abandoning of DTW using LB_Keogh
4. Reordering Early Abandoning

Early Abandoning of LB_Keogh

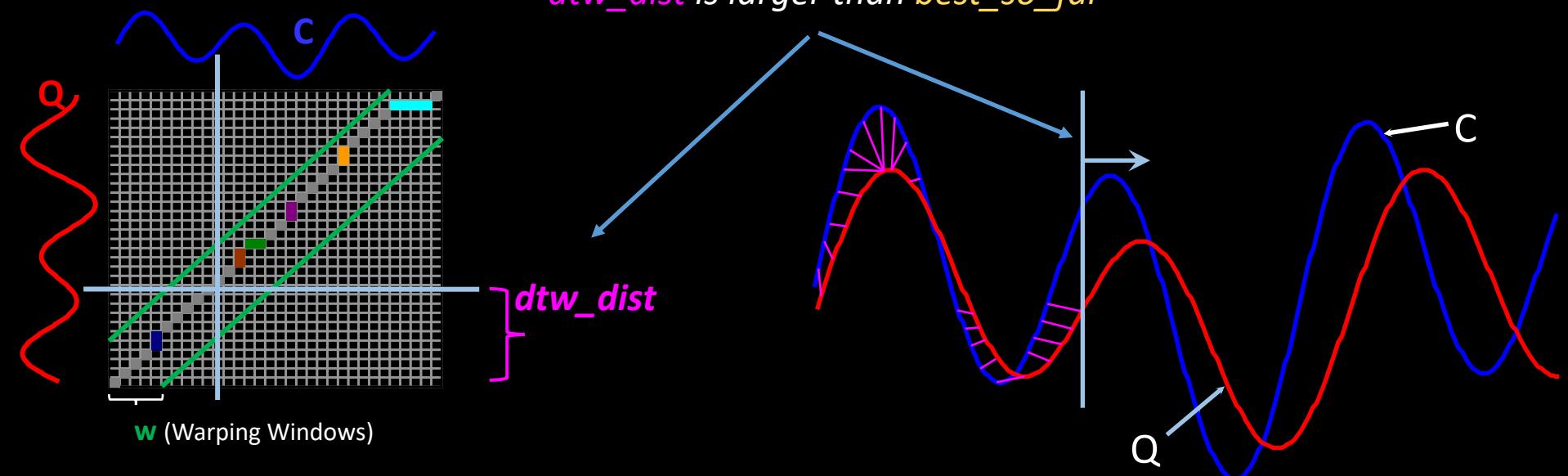
*Abandon the computation, when the accumulated error is larger than **best_so_far***



U, L are upper and lower envelopes of Q

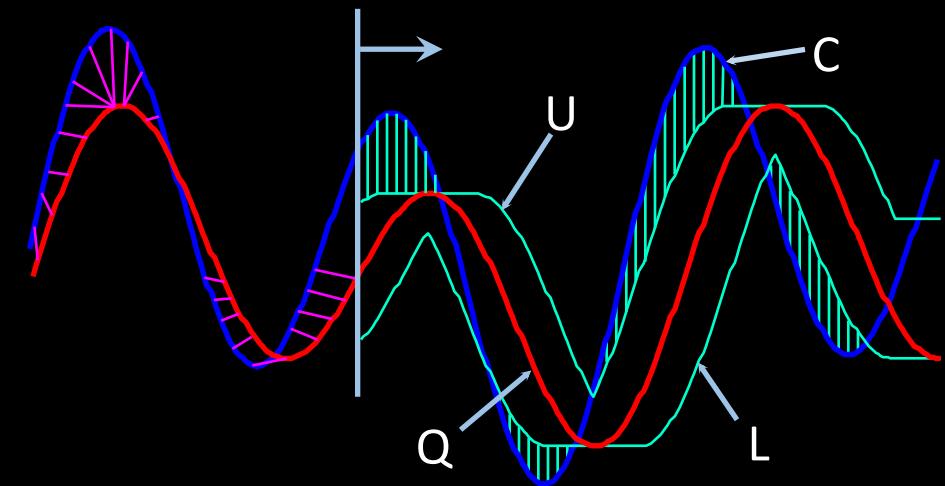
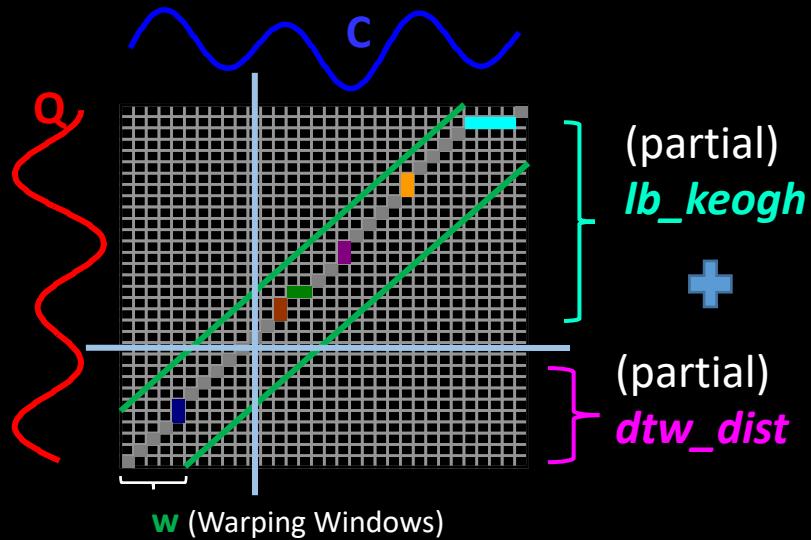
Early Abandoning of DTW

*Abandon the computation, when the
dtw_dist is larger than best_so_far*



Earlier Early Abandoning of DTW using LB_Keogh

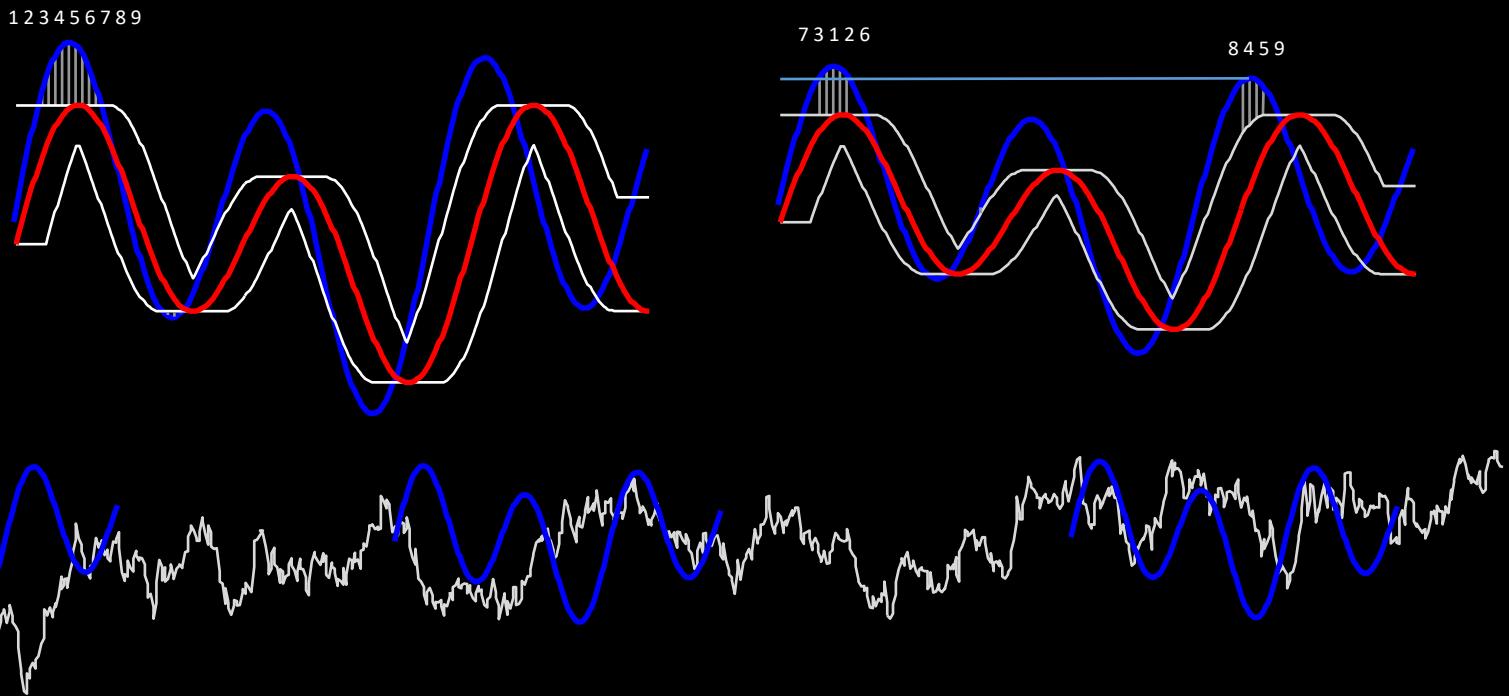
Abandon the computation, when the $dtw_dist + lb_keogh$ is larger than best_so_far



U, L are upper and lower envelopes of Q

Reordering Early Abandoning

- We don't have to compute LB from left to right.
- Order points by expected contribution.



- Order by the absolute height of the query point.

Summary of the techniques

Group-1 Techniques

- Early Abandoning of LB_Keogh
- Early Abandoning of DTW
- Earlier Early Abandoning of DTW using LB_Keogh



Group-2 Techniques

- Just-in-time Z-normalizations
- Reordering Early Abandoning
- Reversing LB_Koegh
- Cascading Lower Bounds



UCR Suite

Code and data is available at:

www.cs.ucr.edu/~eamonn/UCRsuite.html

Experimental Result: Random Walk

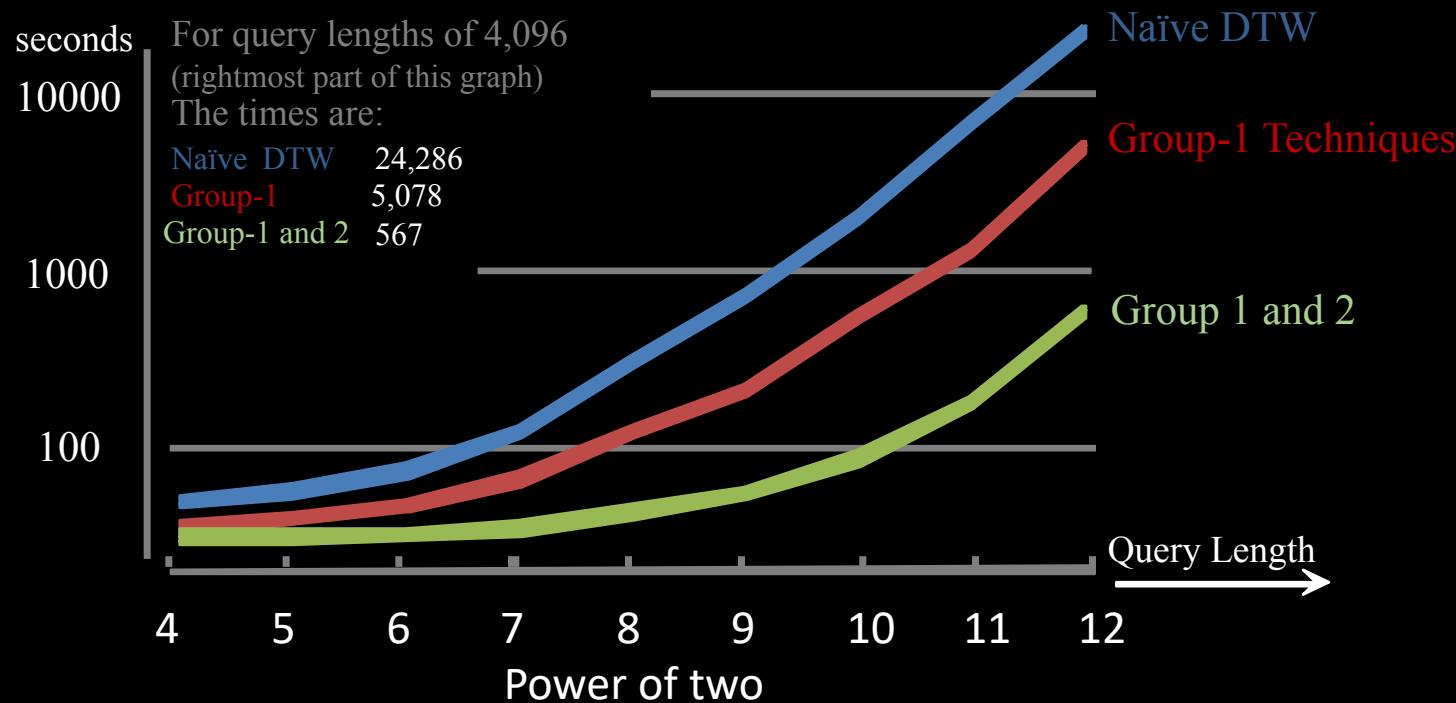
- Random Walk: Varying size of the data, $|Q| = 128$

	Million (Seconds)	Billion (Minutes)	Trillion (Hours)	Extrapolated
DTW-Naive	75.21	1,252.2	20,869	
Group-1	2.447	38.14	472.80	
Group-1 and 2	0.159	1.83	34.09	

- Experiments performed on a commodity machine
- Disk is accessed sequentially as the algorithm is invariant to data order
- The computation is performed while the next disk block is read

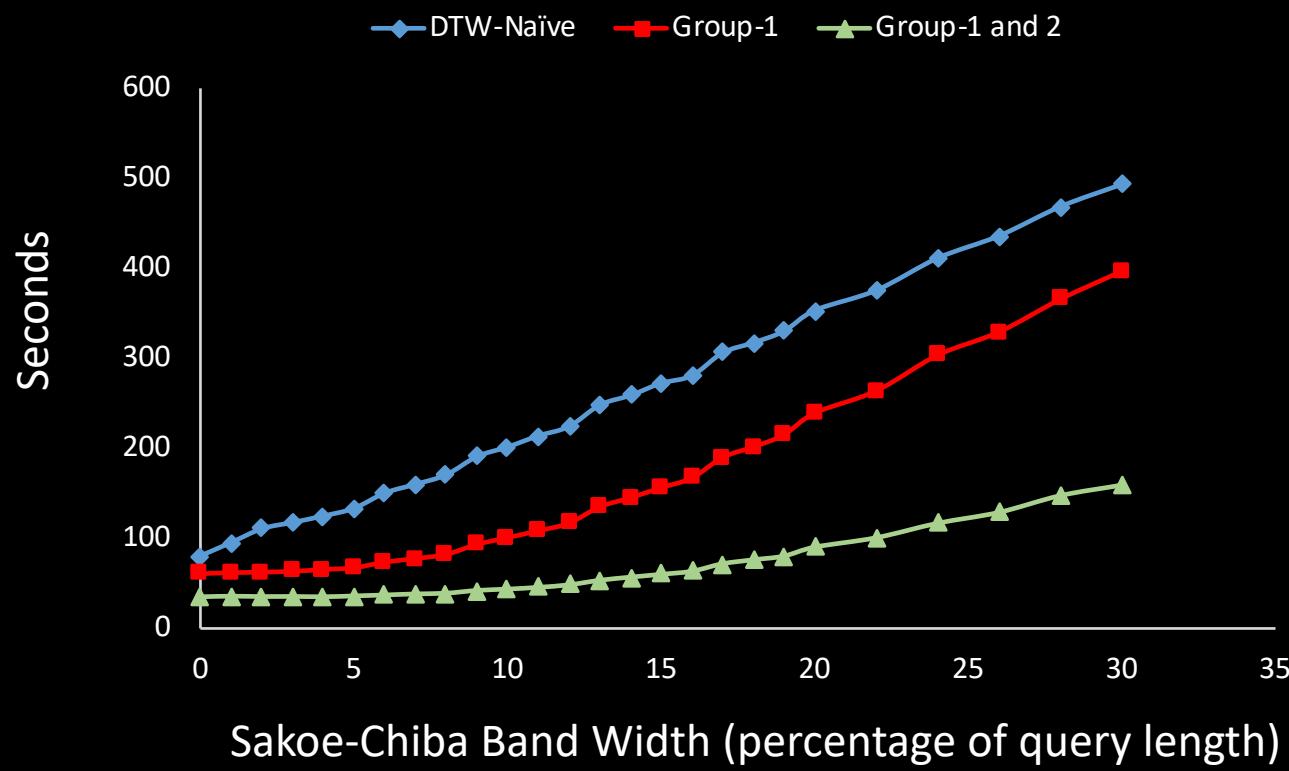
Experimental Result: Random Walk

- Random Walk: Varying size of the query, $n = 20$ million



Experimental Result: Random Walk

- Random Walk: Varying size of the band



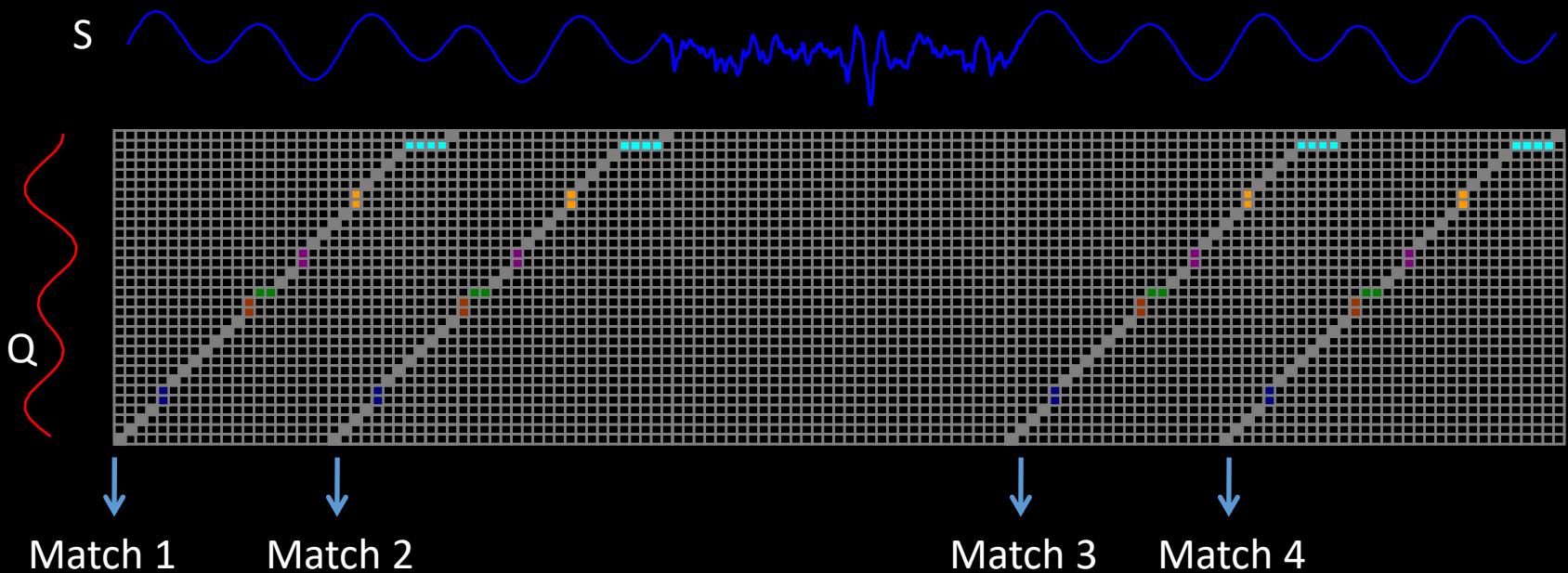
Nearest Subsequence Search

- A **Q** query is given
- A long time series of length n
- $O(n)$ distance calculations are performed to

Find THE nearest subsequence of
the given query under DTW.

Time Warping Subsequence Search

Reuses computation for subsequence matching



- For every new observation only one column is added on the right
- No need for any of the techniques

Normalization is required

- If each window is normalized separately, reuse of computation is no longer possible
- To take advantage of the **bounding** and **abandoning** techniques, we need **just-in-time normalization** with constant overhead per comparison

Just-in-time Normalization

- In one pass, calculate cumulative sums of over x and x^2 and store $C = \sum x$ $C^2 = \sum x^2$
- Subtract two cumulative sums to obtain the sum over a window $S_i = C_{i+w} - C_i$ $S_i^2 = C_{i+w}^2 - C_i^2$
- Use the sums to calculate the means and standard deviations of all windows in linear time

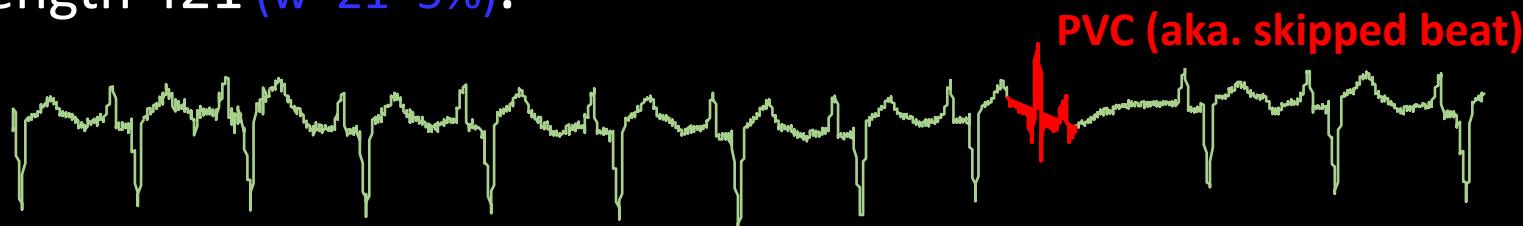
$$\mu_i = \frac{S_i}{w} \quad \sigma_i = \sqrt{\frac{S_i^2}{w} - \left(\frac{S_i}{w}\right)^2}$$

- Dynamically normalize observations when calculating distance and possibly abandon early

$$cost = \left(\frac{x_{ij} - \mu_{xi}}{\sigma_{xi}} - \frac{q_i - \mu_{qi}}{\sigma_{qi}} \right)^2$$

Experimental Result: ECG

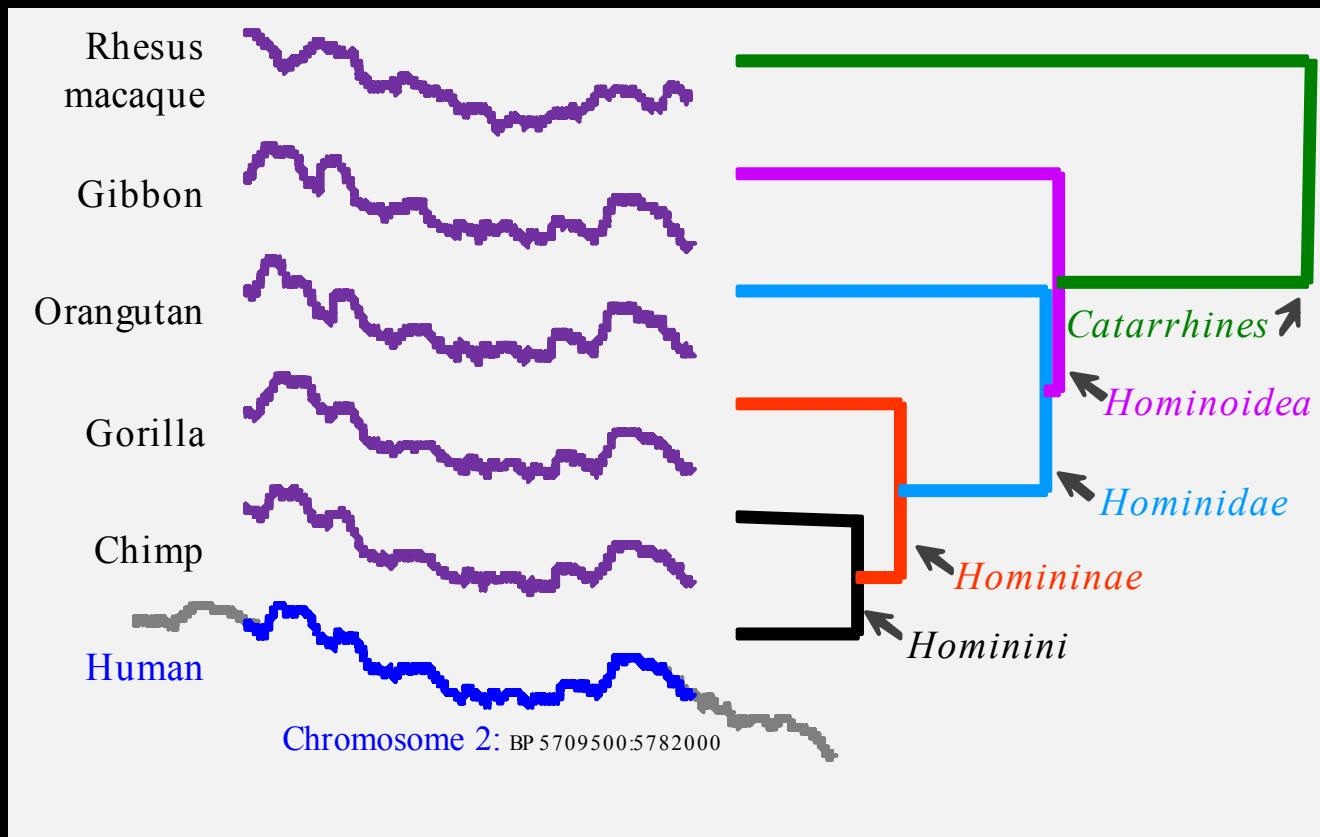
- Data: One year of Electrocardiograms 8.5 billion data points.
- Query: Idealized Premature Ventricular Contraction (PVC) of length 421 ($w=21=5\%$).



	Group-1	Group 1 & 2
ECG	49.2 hours	18.0 minutes

Experimental Result: DNA

- Query: Human Chromosome 2 of length **72,500 bps**
- Data: Chimp Genome 2.9 billion bps
- Time: UCR Suite 14.6 hours

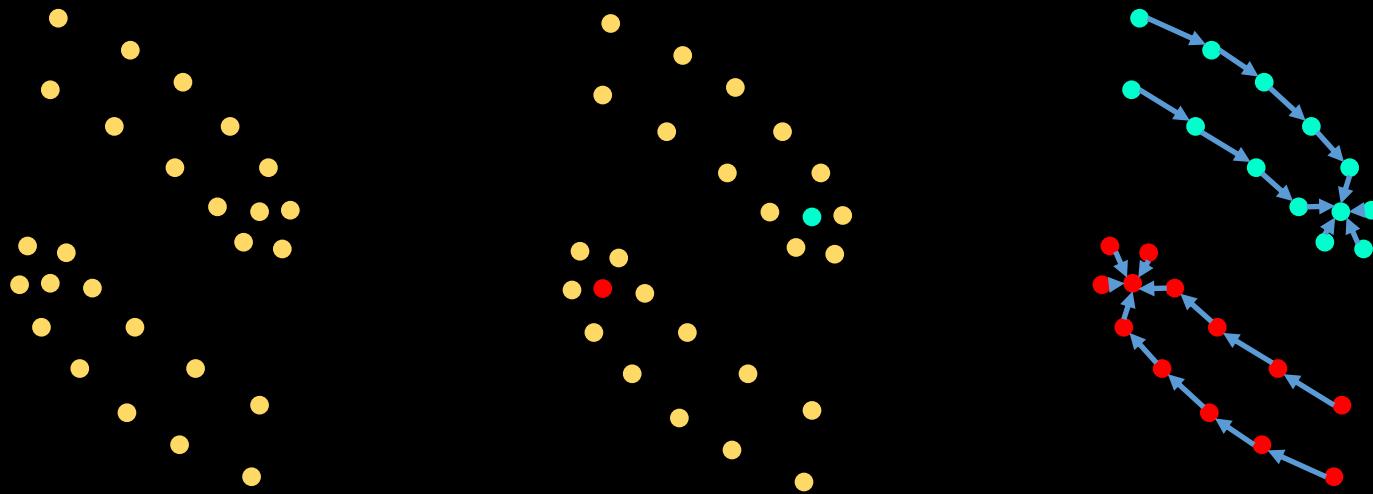


What can be made fast?

- One-to-One comparison
 - Exact Implementation and Constraints
 - Efficient Approximation
 - Exploiting Sparsity
- One-to-Many comparisons
 - Nearest Neighbor Search
 - In a database of independent time series
 - In subsequences of a long time series
 - Density Estimation
 - In clustering
 - Averaging Under Warping
 - In classification
- Many-to-Many comparisons
 - All-pair Distance Calculations

Density based clustering

- Density Peaks (DP)* Algorithm
 - Find the densities of every point to pick cluster centers
 - Connect every point to the nearest higher density point



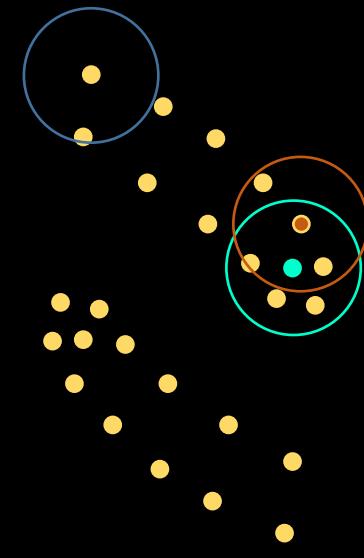
*Rodriguez, A., & Laio, A. (2014). *Clustering by fast search and find of density peaks*. *Science*, 344(6191), 1492–1496.

Range Search/Density Estimation

- Density is estimated by the number of points within a radius/threshold t

Algorithm Bounding_Range_Search(Q, t)

```
1. for all sequences  $C_i$  in database
2.   LB_dist = lower_bound_distance ( $C_i, Q$ )
3.   if LB_dist <  $t$ 
4.     UB_dist = upper_bound_distance ( $C_i, Q$ )
5.     if UB_dist <  $t$  then output  $C_i$ 
6.   else
7.     true_dist = DTW( $C_i, Q$ )
8.     if true_dist <  $t$ 
9.       output  $C_i$ 
10.    endif
11.  endif
12. endif
13. endfor
```



Try to use an upper bound to identify a point within the range

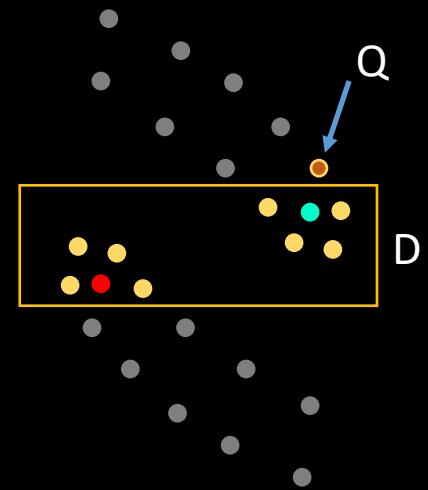


Density Connectedness

- Distance between a pair of points is an upper bound of the NN distance from both of the points

Algorithm Bounding_Scan(D, Q)

```
1. best_so_far = min(upper_bound_NN_distance( $D, Q$ ))
2. for all sequences in  $D$ 
3.   LB_dist = lower_bound_distance( $C_i, Q$ );
4.   if LB_dist < best_so_far
5.     true_dist = DTW( $C_i, Q$ );
6.     if true_dist < best_so_far
7.       best_so_far = true_dist;
8.       index_of_best_match = i;
9.     endif
10.    endif
11.  endfor
```

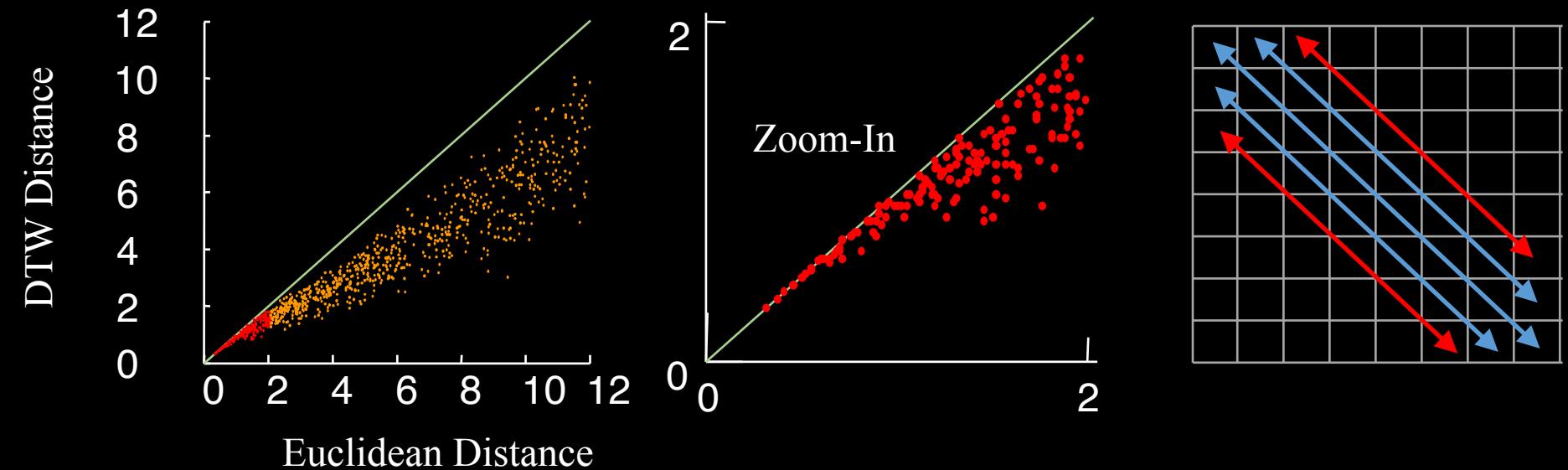


Try to use an upper bound to the NN distance as the *best_so_far*



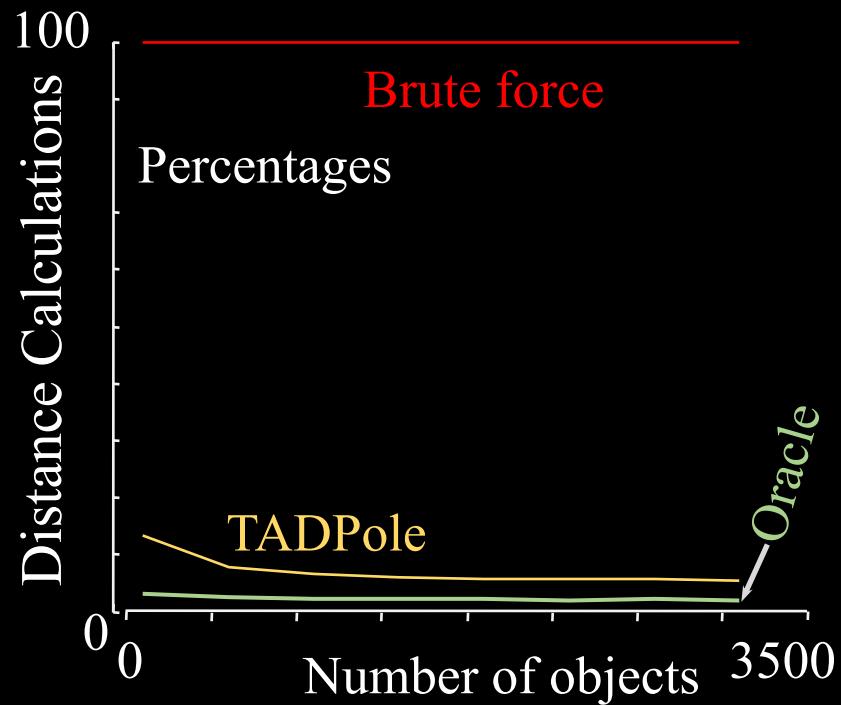
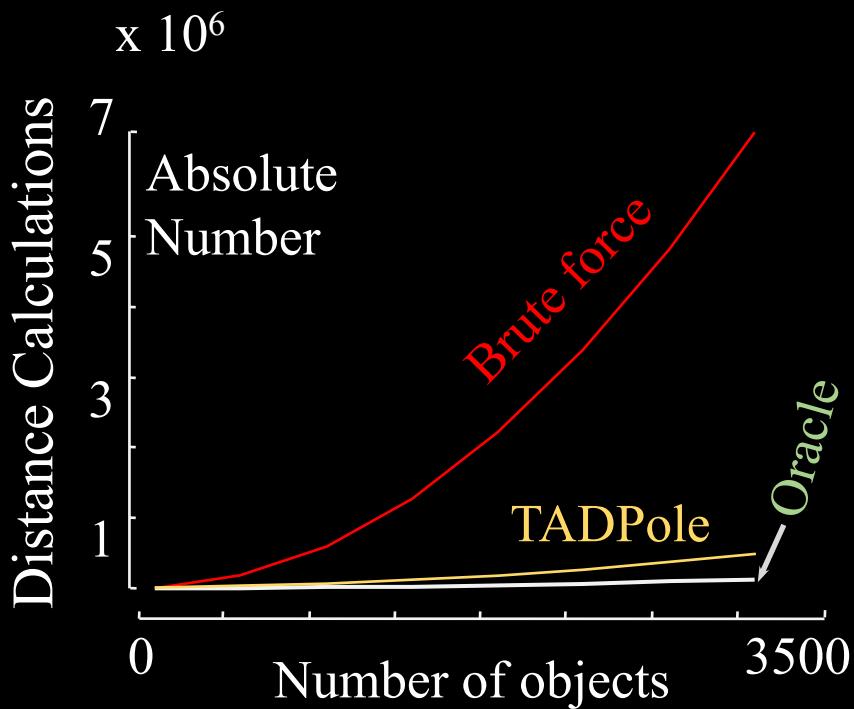
Upper bounding

- Euclidean distance is a trivial upper bound
- DTW distance in a band w is an upper bound for DTW distance in band $w+1$



Speedup by upper bounds

Density Peak: 9 Hours
TADPole: 9 minutes



StarLightCurves dataset

What can be made fast?

- One-to-One comparison
 - Exact Implementation and Constraints
 - Efficient Approximation
 - Exploiting Sparsity
- One-to-Many comparisons
 - Nearest Neighbor Search
 - In a database of independent time series
 - In subsequences of a long time series
 - Density Estimation
 - In clustering
 - Averaging Under Warping
 - In classification
- Many-to-Many comparisons
 - All-pair Distance Calculations

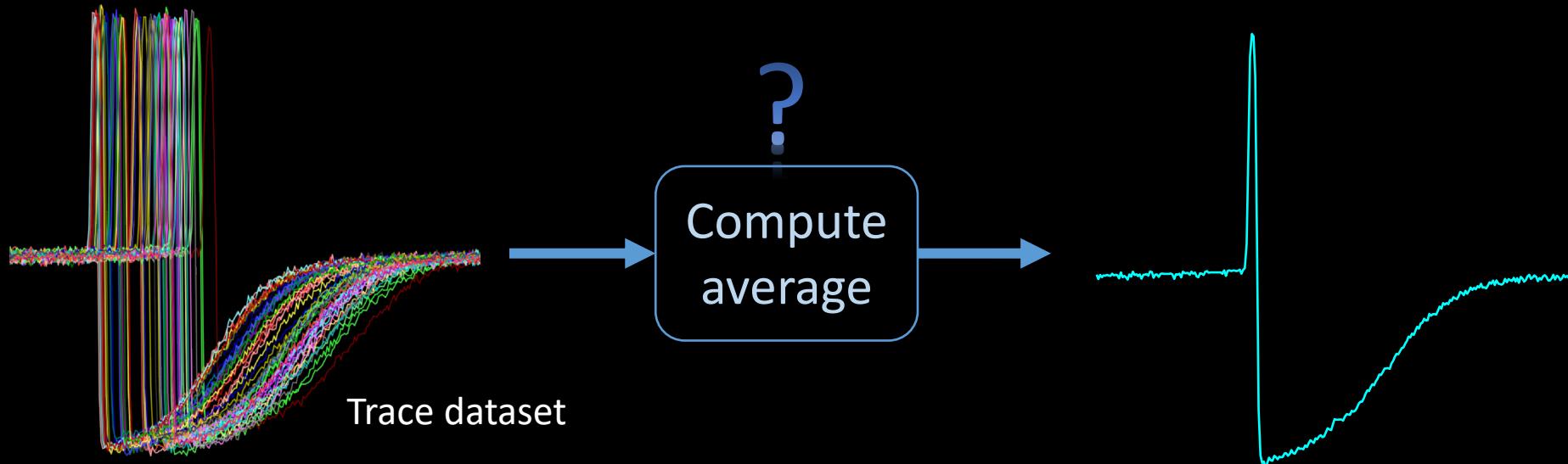
Data Reduction for 1NN Classification

- The training set is reduced to a smaller set keeping a representative set of labeled instances
- Smaller training set entails performance gain
- Smaller training set may gain accuracy if noisy instances are filtered effectively
- **Reduction methods**
 - Random Selection
 - Rank the instances and take top-K
 - Cluster instances based on proximity and take representative from each cluster

Many clustering algorithms require finding a centroid of two or more instances

The issue is then:

- *How to average time series consistently with DTW?*



Mathematically, the mean \bar{o} of a set of objects O embedded in a space induced by a distance d is:

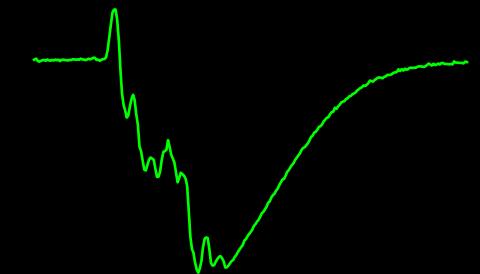
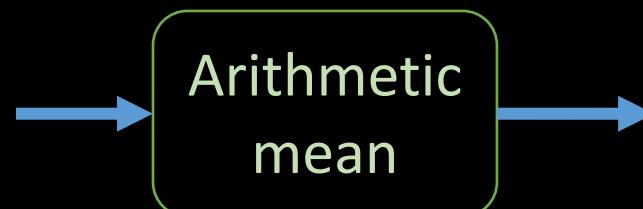
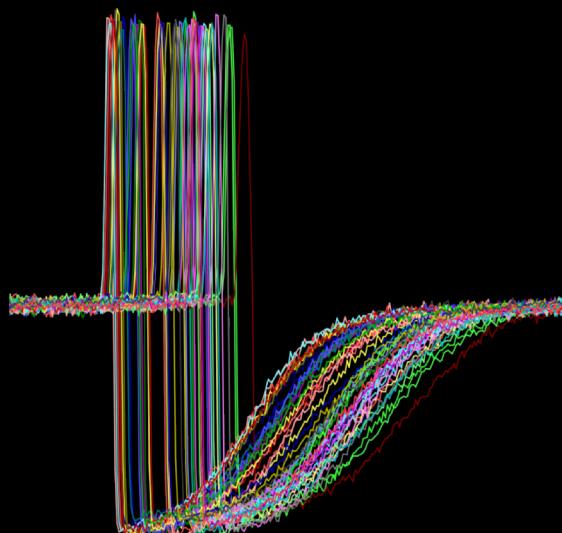
$$\arg \min_{\bar{o}} \sum_{o \in O} d^2(\bar{o}, o)$$

The mean of a set minimizes the sum of the squared distances

The *arithmetic mean* solves the problem exactly

$$\bar{o} = \frac{1}{N} \sum_{o \in O} o$$

If d is the Euclidean distance



To solve the optimization problem for DTW distance, we need to perform simultaneous alignment of many time series.

But, finding the optimal multiple alignment:

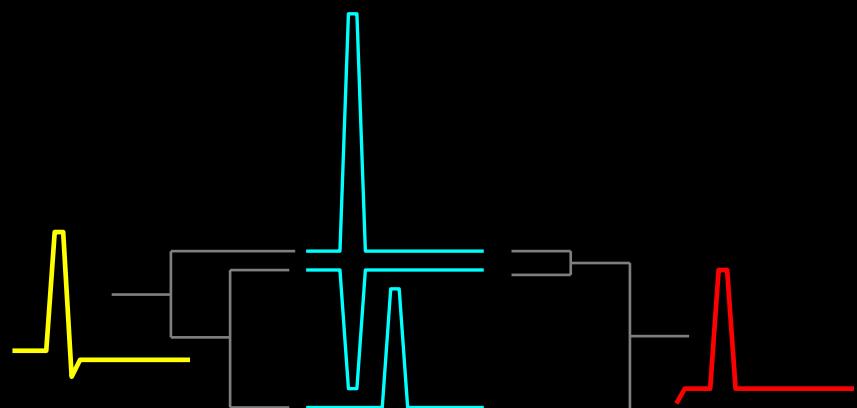
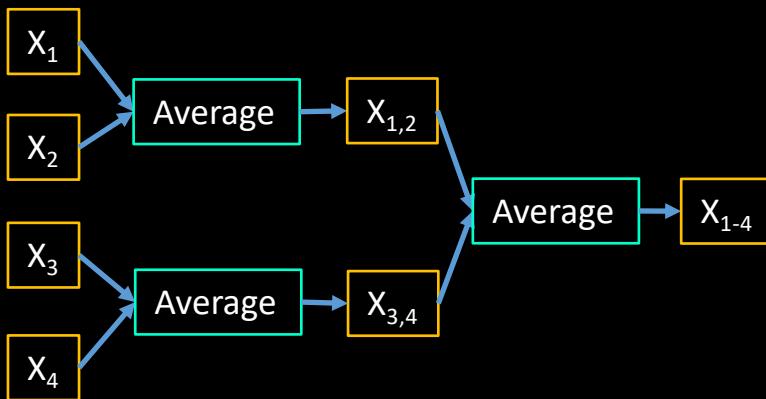
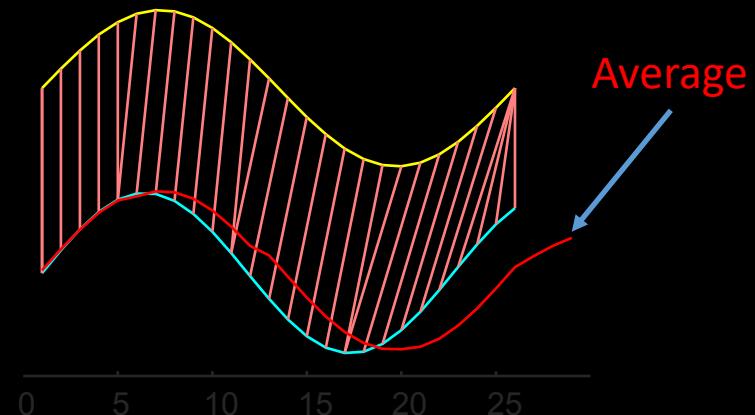
1. Is **NP-complete** [a]
2. Requires $O(L^N)$ operations
 - L is the length of the sequences (≈ 100)
 - N is the number of sequences ($\approx 1,000$)

⇒ Efficient solutions will be heuristic

- Pairwise Averaging
- DTW Barycenter Averaging (DBA)

Pairwise averaging for DTW

- Average each alignment between the two time series
- Commonly increases the length
- Chaining can produce average over a set
- The operation is not associative, the average produced depends on the order

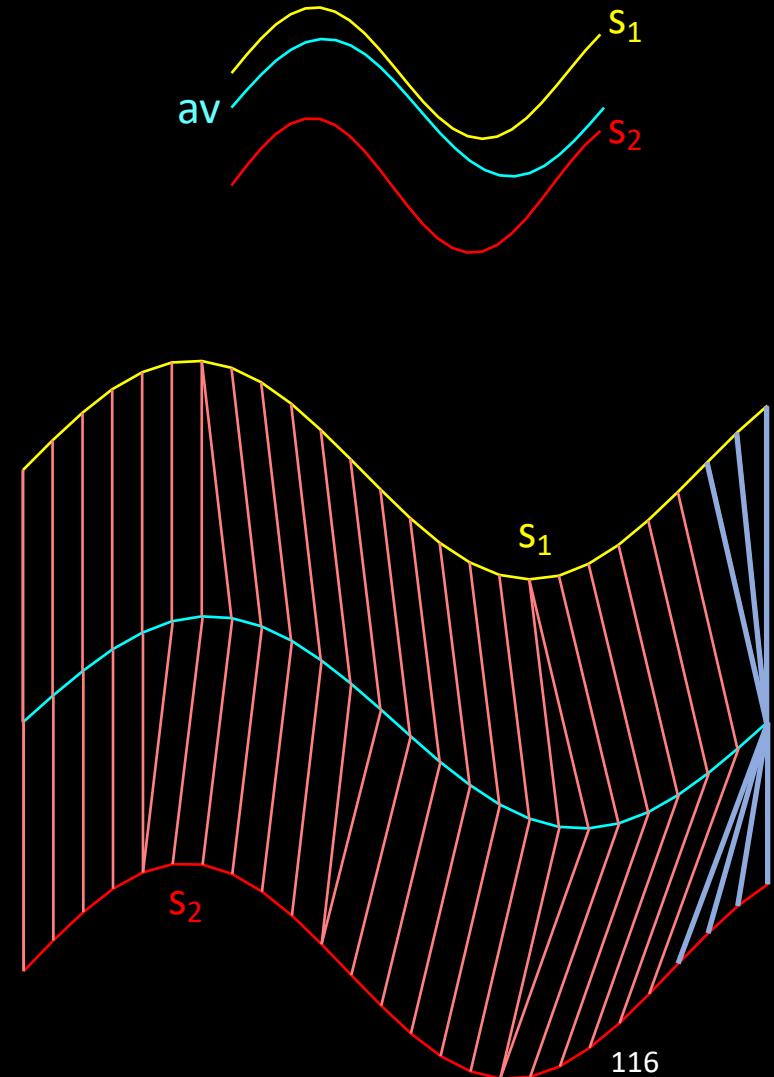


DTW Barycenter Averaging (DBA)

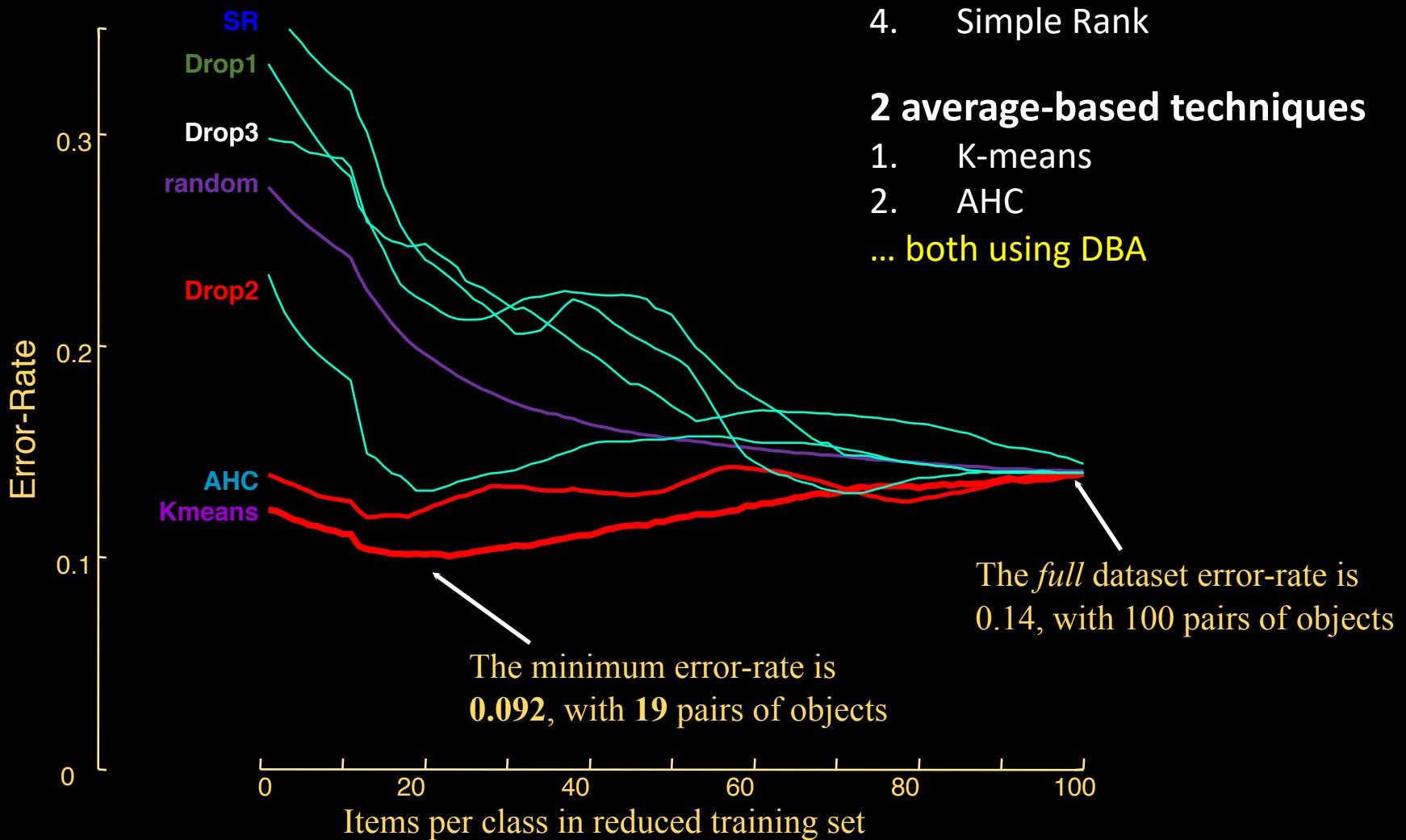
Algorithm DBA(D,av)

```
1 | Iterate until convergence  
2 |   for each sequence  $s_i$  in D  
3 |      $A_i = \text{GetAlignment}(\text{DTW}(s_i, \text{av}))$   
4 |   for each observation  $j$  in av  
5 |      $\text{av}[j] = \text{mean}([A_1[j] A_2[j] A_3[j] \dots A_n[j]])$ 
```

- Does not increase length
- Does not depend on the order of the points



Experimental Evaluation on Insect Data



What can be made fast?

- One-to-One comparison
 - Exact Implementation and Constraints
 - Efficient Approximation
 - Exploiting Sparsity
- One-to-Many comparisons
 - Nearest Neighbor Search
 - In a database of independent time series
 - In subsequences of a long time series
 - Density Estimation
 - In clustering
 - Averaging Under Warping
 - In classification
- Many-to-Many comparisons
 - All-pair Distance Calculations

Speeding up DTW: many-to-many

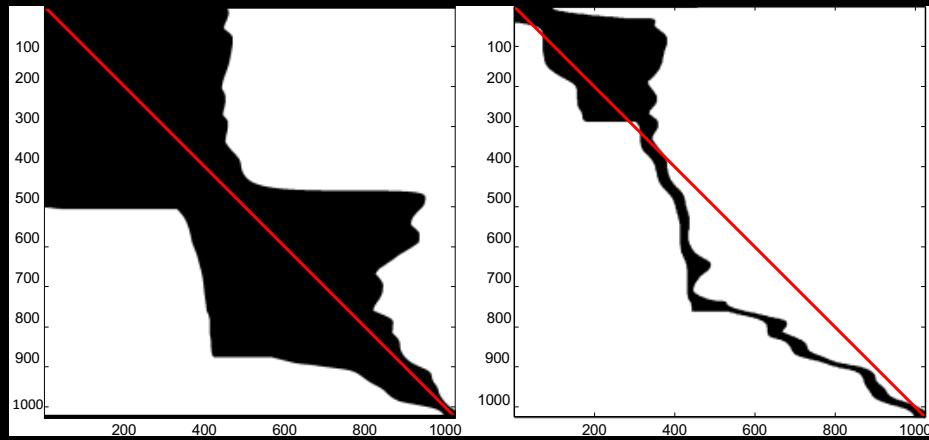
- Several variants
 - Self-join within a threshold - top-K Self-join
 - Use similarity search techniques as subroutine
 - Application: Motif discovery [a], Discord discovery
 - A/B Join within a threshold - top-K A/B Join
 - Use similarity search techniques as subroutine
 - Application: Motion Stitching [b]
 - All-pair distance matrix
 - Use techniques to speedup one-to-one comparisons
 - Application: Hierarchical Clustering

[a] N Chavoshi, H Hamooni, A Mueen, "DeBot: Real-Time Bot Detection via Activity Correlation" UNM Technical Report

[b] Y. Chen, G. Chen, K. Chen and B. C. Ooi, "Efficient Processing of Warping Time Series Join of Motion Capture Data," ICDE, 2009, pp. 1048-1059.

PrunedDTW: speeding up all-pair distance matrix calculation

- Two types of pruning when calculating DTW matrix
- Exact method



UB = Euclidean distance

UB = DTW distance

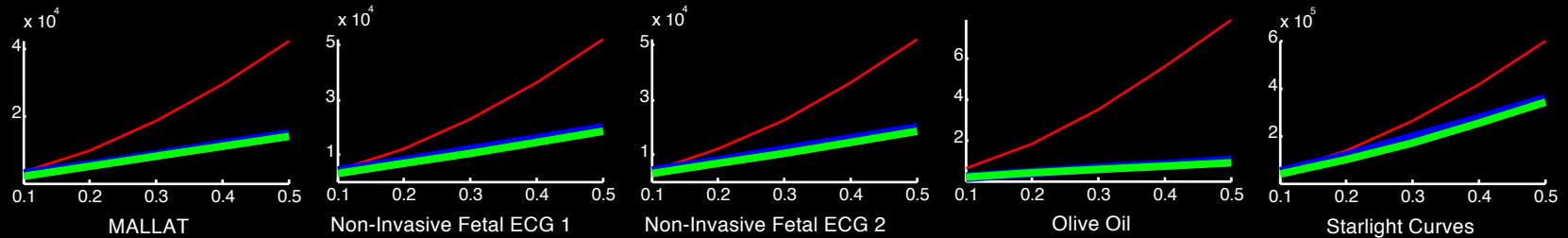
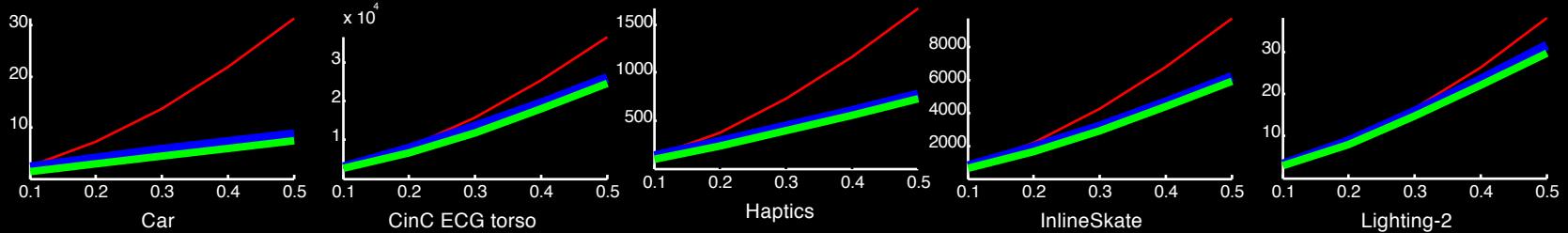
	0	1	2	3	4	5	6
3	∞			$\leq UB$			
4	∞	$> UB$	$> UB$	$\leq UB$	$\leq UB$		
5	∞	$\leq UB$					
6	∞						$\leq UB$

Lower triangle pruning

	0	1	2	3	4	5	6
0	0	∞	∞	∞	∞	∞	∞
1	∞	$\leq UB$	$\leq UB$	$\leq UB$	$> UB$	$> UB$	$> UB$
2	∞		$\leq UB$	$\leq UB$	$> UB$	$\leq UB$	
3	∞			$\leq UB$			

Upper triangle pruning

Experiments



Time (s)
↑
Warping Window Length

— DTW

— Pruned
DTW

— Oracle
DTW

Conclusion

- Nearest neighbor search under warping is fast enough for most practical purposes
- Data reduction improves 1NN classification both in speed and accuracy
- DTW is an extraordinarily powerful and useful tool. Its uses are limited only by our imagination.



Additional References

- H. Sakoe and S. Chiba, “Dynamic programming algorithm optimization for spoken word recognition,” IEEE Trans. Acoust. Speech, Lang. Process., vol. 26, no. 1, pp. 43–50, 1978.
- Mustafa S. Çetin, Abdullah Mueen, Vince D. Calhoun: Shapelet Ensemble for Multi-dimensional Time Series. SDM 2015: 307-315
- X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh, “Experimental comparison of representation methods and distance measures for time series data,” Data Min. Knowl. Discov., vol. 26, no. 2, pp. 275–309, 2013.
- D. Sart, A. Mueen, W. Najjar, V. Niennattrakul, and E. Keogh, “Accelerating Dynamic Time Warping Subsequence Search with GPUs and FPGAs. ICDM 2010,” in Proceedings - IEEE International Conference on Data Mining, ICDM, 2010, pp. 1001–1006.
- Abdullah Mueen, Hossein Hamooni, Trilce Estrada: Time Series Join on Subsequence Correlation. ICDM 2014: 450-459
- Ira Assent, Marc Wichterich, Ralph Krieger, Hardy Kremer, and Thomas Seidl. 2009. Anticipatory DTW for efficient similarity search in time series databases. Proc. VLDB Endow. 2, 1 (August 2009), 826-837.