Similarity Search for Time Series Subsequence under Dynamic Time Warping

Dissertation Presentation for MSc in Information Technology, PolyU

NG Yiu Wai

May 2020

Table of Content

Introduction – Why Time Series

- Searching similar time series data is a common problem in many application.
- Similarity of subsequence and query is measured by distance functions.
- Dynamic time warping is one of the best measurement of similarity, because some data are warped by nature.

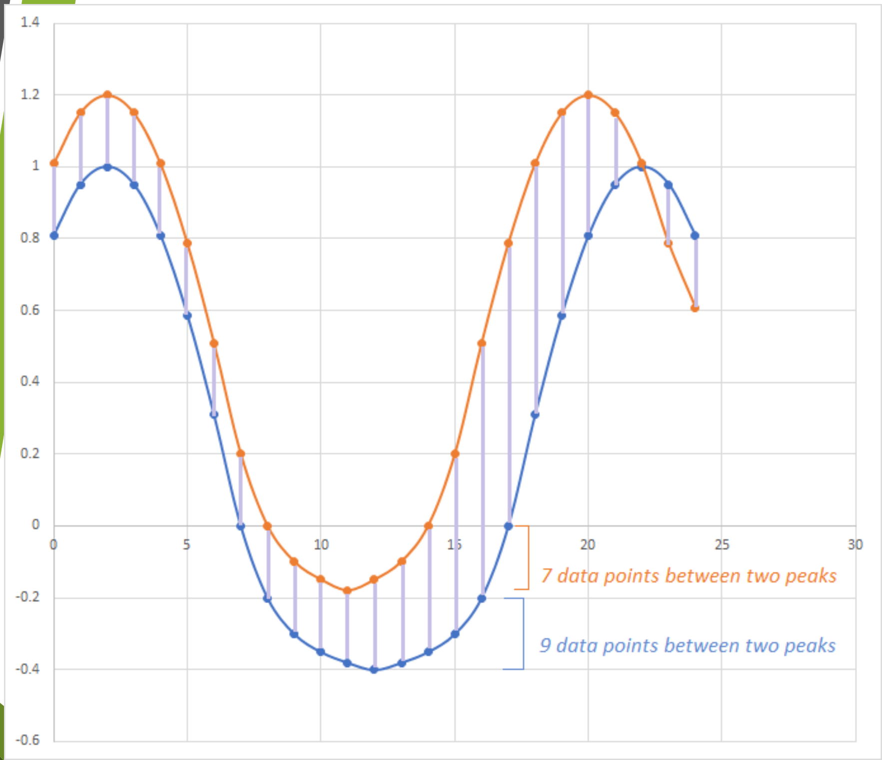
Examples of Time Series Data

- Financial data: sampling the price at regular time intervals
- e.g. user is interested in finding two consecutive peaks. Interval between peaks are not important.

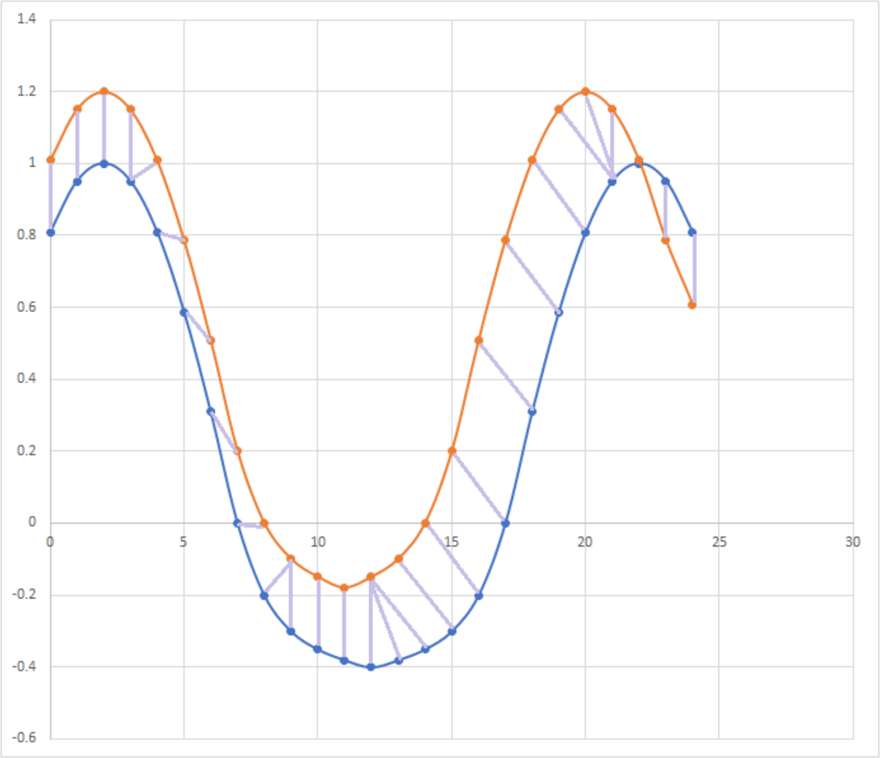
Examples of Time Series Data

- Example of two consecutive peaks :

Euclidean Distance

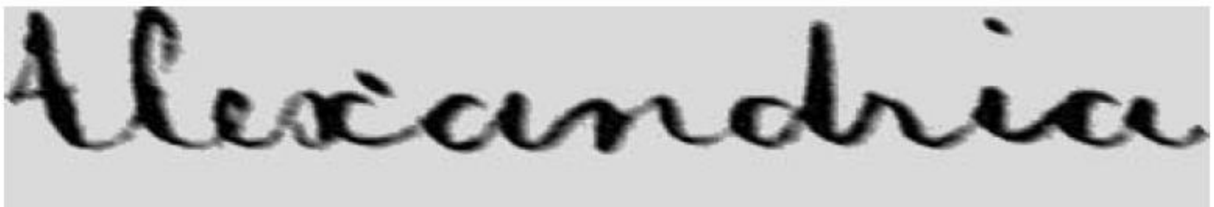


Dynamic Time Warping

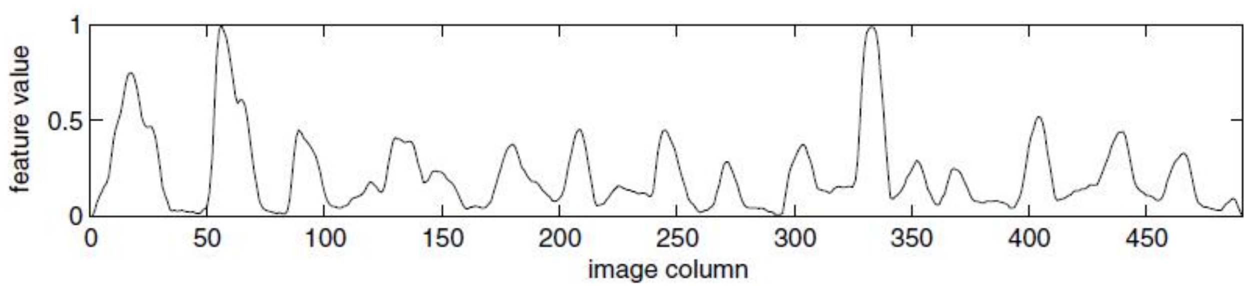


Examples of Time Series Data

- Text recognition: transforming image to time series data
- e.g. We are interested in the letters.
  Spaces between letters are not important.



(a) original image: slant/skew/baseline-normalized, cleaned.

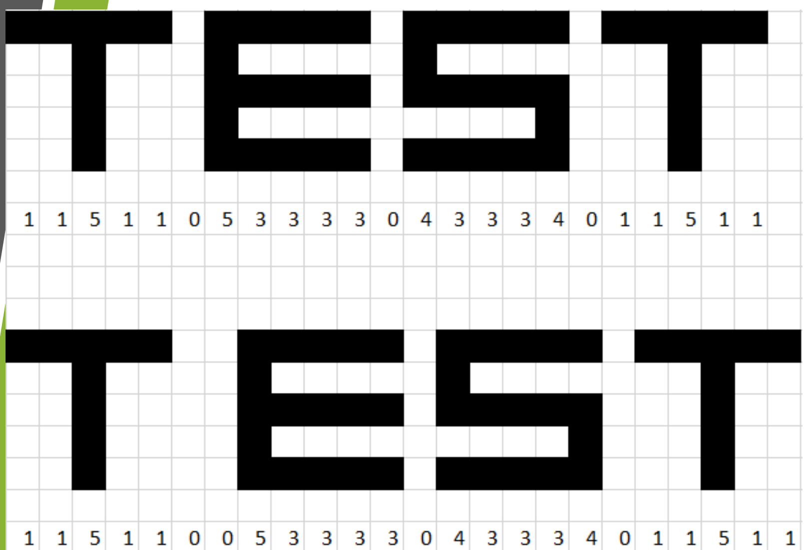


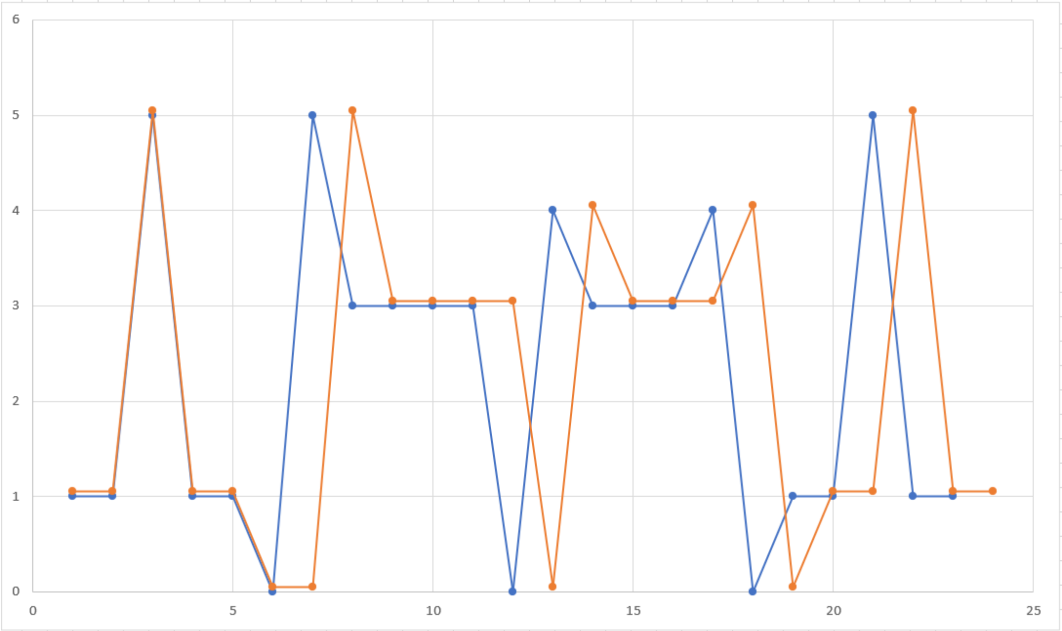
(b) normalized projection profile.

Source: Rath, Toni M., and Raghavan Manmatha. "Word image matching using dynamic time warping." 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings. Vol. 2. IEEE, 2003.

Examples of Time Series Data

- Example of text with spacing:

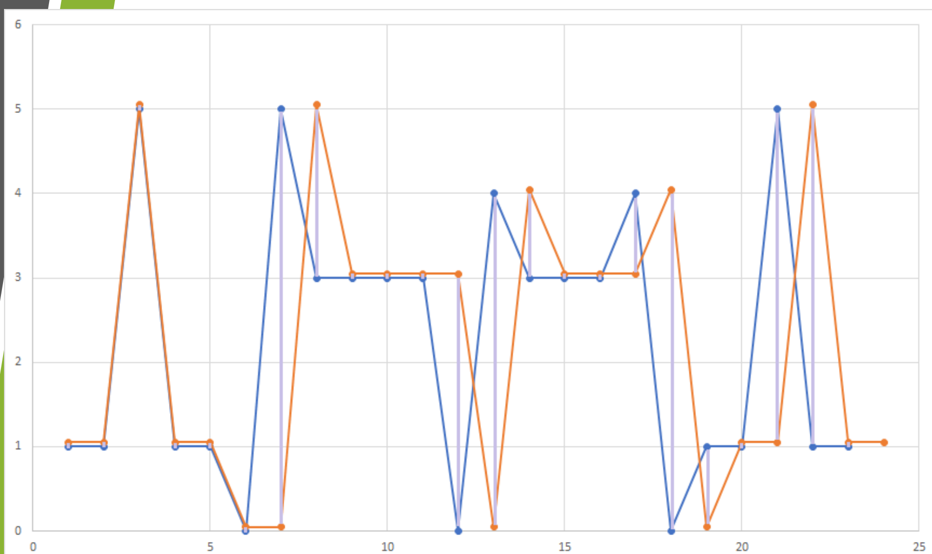




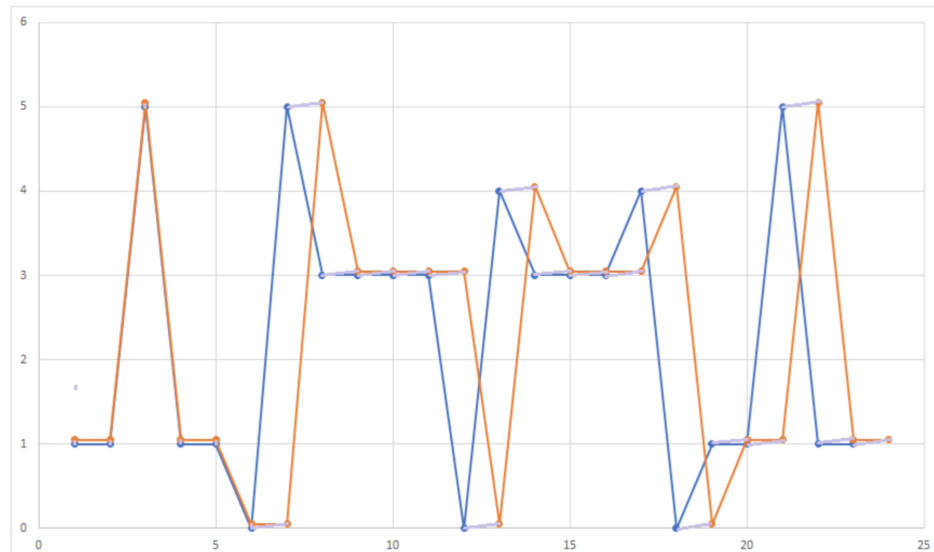
Examples of Time Series Data

- Example of text with spacing:

Euclidean Distance



Dynamic Time Warping

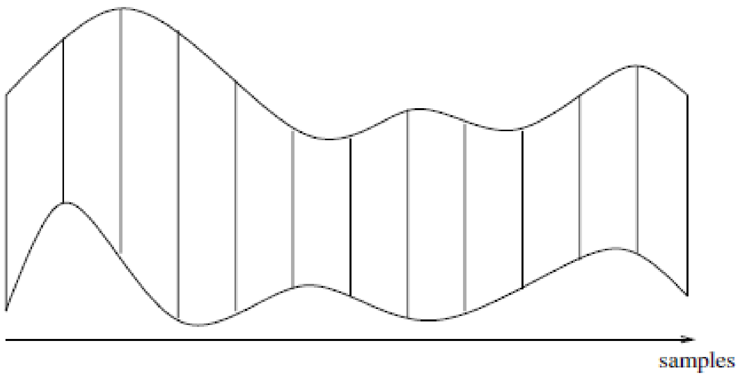


Similarity of Time Series Data

Euclidean Distance

- **Definition**: $\sqrt{\sum_{i=1}^{N}(q_i - s_i)^2}$

  for a query of length = *N*.



- Time complexity: $O(N)$
- Simple to compute
- Cannot fit our needs if the subsequences is warped.

Similarity of Time Series Data

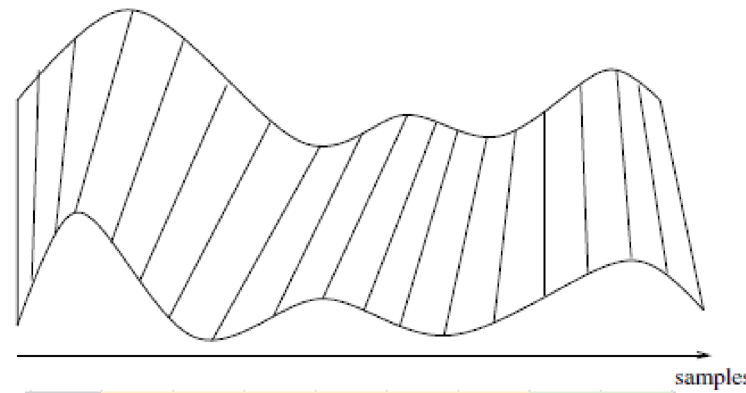
<u>Dynamic Time Warping</u>

- **Definition**:

$$DTW = \min\left(\sqrt{\sum_{k=1}^{K} w_k}\right) \text{ for } w_k = \left(q_i - s_j\right)^2$$

- The path $W = w_1, w_2, \ldots, w_K$ is subjected to several constraints:
  1) Boundary conditions: $w_1 = (1,1)$ and $w_K = (N, N)$
  2) Continuity: Given $w_k = (a, b)$, then $w_{k-1} = (a', b')$, where $a - a' \leq 1$ and $b - b' \leq 1$.
  3) Monotonicity: Given $w_k = (a, b)$, then $w_{k-1} = (a', b')$, where $a - a' \geq 0$ and $b - b' \geq 0$.

- Time complexity: $O(N^2)$
- Finding shortest path in a $N*N$ matrix for a query of length = N.



-1.2	3.1	5.9	4.4	3.1	2.1	1.2	0.3	0
0.8	0.1	0.1	0	0.1	0.4	0.9	6.8	5.2
0.8	0.1	0.1	0	0.1	0.4	0.9	6.8	5.2
1.4	0.7	0	0.2	0.7	1.3	2.2	9.8	7.8
0.8	0.1	0.1	0	0.1	0.4	0.9	6.8	5.2
-0.7	1.6	3.6	2.5	1.6	0.8	0.3	1.1	0.5
-1.2	3.1	5.9	4.4	3.1	2.1	1.2	0.3	0
-0.7	1.6	3.6	2.5	1.6	0.8	0.3	1.1	0.5
	0.5	1.2	0.9	0.5	0.2	-0.1	-1.8	-1.4

Similarity of Time Series Data

Dynamic Time Warping

- Sakoe-Chiba band is a global constraint for dynamic time warping.
- It can avoid a relatively small section of one sequence maps onto a relatively large section of another sequence.
- If width of Sakoe-Chiba band is *R*, then the time complexity is $O(N*(2R+1))$.
- A better distance function for measuring similarity of time series data

Finding Shortest Path for *R* = 2

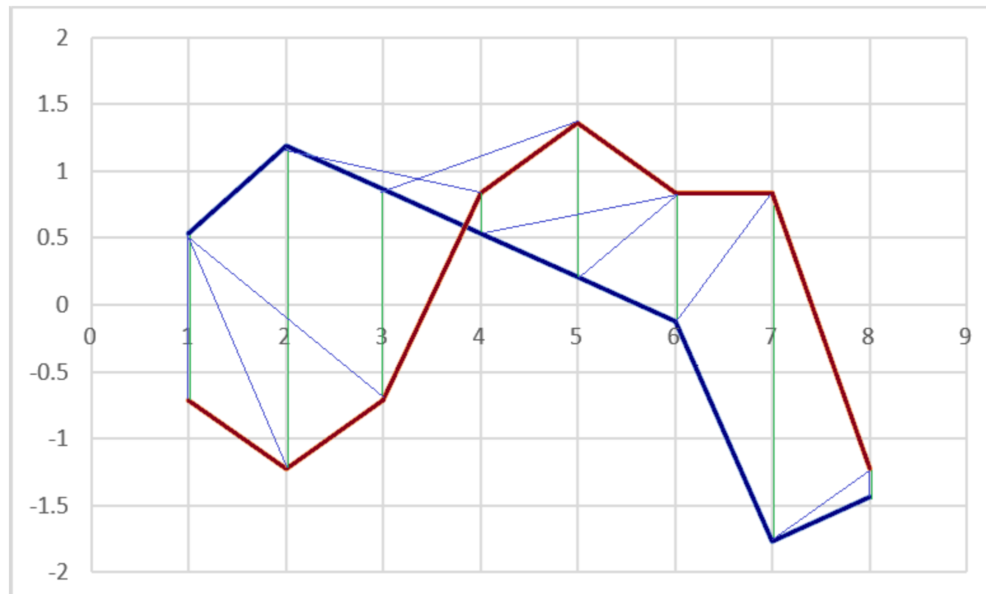
-1.2	3.1	5.9	4.4	3.1	2.1	1.2	0.3	0
0.8	0.1	0.1	0	0.1	0.4	0.9	6.8	5.2
0.8	0.1	0.1	0	0.1	0.4	0.9	6.8	5.2
1.4	0.7	0	0.2	0.7	1.3	2.2	9.8	7.8
0.8	0.1	0.1	0	0.1	0.4	0.9	6.8	5.2
-0.7	1.6	3.6	2.5	1.6	0.8	0.3	1.1	0.5
-1.2	3.1	5.9	4.4	3.1	2.1	1.2	0.3	0
-0.7	1.6	3.6	2.5	1.6	0.8	0.3	1.1	0.5
	0.5	1.2	0.9	0.5	0.2	-0.1	-1.8	-1.4

Euclidean Distance & Dynamic Time Warping

Euclidean Distance: 19.11

Dynamic Time Warping: 8.35

-1.2	3.1	5.9	4.4	3.1	2.1	1.2	0.3	0
0.8	0.1	0.1	0	0.1	0.4	0.9	6.8	5.2
0.8	0.1	0.1	0	0.1	0.4	0.9	6.8	5.2
1.4	0.7	0	0.2	0.7	1.3	2.2	9.8	7.8
0.8	0.1	0.1	0	0.1	0.4	0.9	6.8	5.2
-0.7	1.6	3.6	2.5	1.6	0.8	0.3	1.1	0.5
-1.2	3.1	5.9	4.4	3.1	2.1	1.2	0.3	0
-0.7	1.6	3.6	2.5	1.6	0.8	0.3	1.1	0.5
	0.5	1.2	0.9	0.5	0.2	-0.1	-1.8	-1.4



Our application:

The goal of this dissertation is to solve the exact similarity search for normalized arbitrary-length long time series query for large dataset. A few queries would be asked.

It is a common problem for financial data analysis. Investor would like to search similar patterns in prices.

Large Dataset:

Market price data updates for every minute. In FX market, which has around 250 workings days per year, the length of 3-year dataset is about 1 million.

Normalized:

People are interested in studying investment return. Return could be defined as normalized price movement.

Arbitrary-length:

The definition of short-term or long-terms can be varied because of recent market news. For example, short-term can be defined as 2 days when the company announces its financial performance. It can be defined as 5 days during other periods of the year.

Long query:

Query length is long. For example, user may wish to analyze prices in FX market for every minute. FX market operates 24 hours in each day. Query of 1-day prices would have the length of 1,440.

A few queries:

A few queries are asked on the same data set. For example, users are interested in finding subsequences which are matched in short-term, middle-term and long-terms. Three queries are asked.

Large Dataset:

From Jan 2015 to Dec 2019, total 5 years data
About 250 days per year and 1,440 data point per day (price in minute)
i.e. length of dataset = 1,800,000

Long query & Arbitrary-length :

e.g. 1-week query usually has length of 7,200.
But it has length 5,760 if there are only 4 working days in that week.

Also it should be normalized. A few queries should be asked.

Problem Definition:

Given a long time series X, a query sequence Q and a distance threshold ε ($\varepsilon \geq 0$), find all subsequences S of length $|Q|$ from X, which satisfy $D(S', Q')$, where S' and Q' are the normalized series of S and Q respectively.

The distance $D(S', Q')$ is dynamic time warping under constraint of Sakoe-Chiba Band with width R.

All matching subsequences S should be found. It is exact search for normalized subsequences, but not approximate search.

The length of query $|Q|$ is arbitrary. Assumes that user will ask a few queries (e.g. $\leqq$5 queries) for the same dataset.

Our concern:

How to accelerate the calculation of dynamic time warping, which has time complexity of $O(N*(2R+1))$, between normalized query and subsequence?

State-of-Art Method: UCR Suite

- Indexing search is not preferred. It is because the index building cost for arbitrary-length subsequences is high.
- For example, we have a time series sequence of length N. Arbitrary-length means query could have length = 2 to $(N – 1)$.
- The number of subsequence (i.e. number of leaf node) to be indexed is $(N + 1)*(N – 2)/2$.

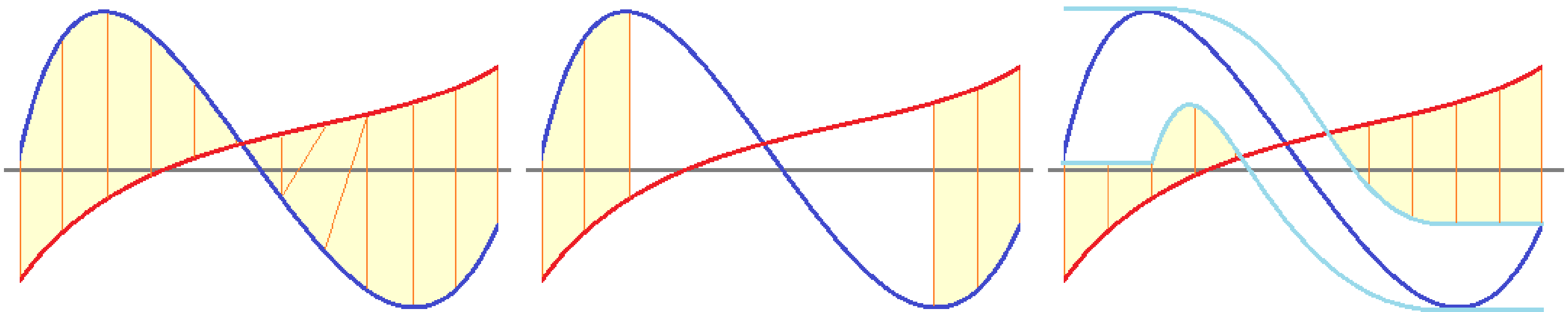
- Sequential search should be used.
- UCR Suite accelerate sequential search by 3 techniques:
    1) Cascading lower bounds
    2) Online z-normalization
    3) Reordering of comparison

Cascading lower bounds

- Lower bound function is defined as some functions that:
  1) $DTW(Q,S) \geqq LB(Q,S)$
  2) $LB(Q,S)$ is fast to compute.
- A good lower bound function should be tight. Ideally, $LB(Q,S)$ should be slight lower than $DTW(Q,S)$ only.
- The two lower bound functions in UCR Suite, namely *LB_Kim* and *LB_Keogh*, are Euclidean distance.
- They effectively transform distance function from dynamic time warping to Euclidean distance.

Cascading lower bounds

- *LB_Kim* and *LB_Keogh*:

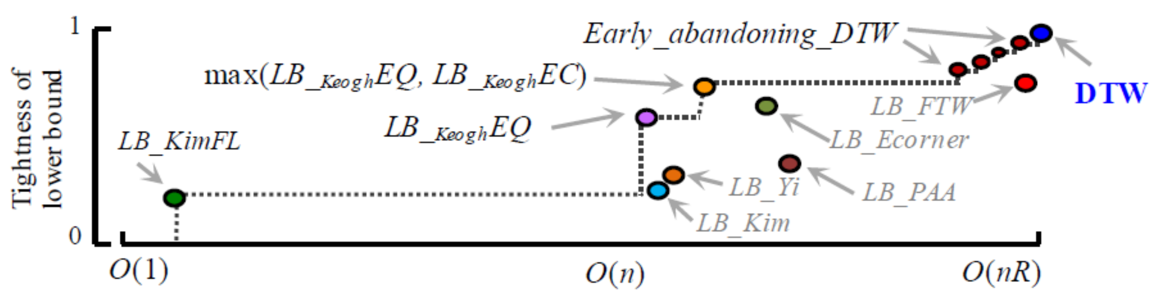


Dynamic Time Warping Distance

LB_Kim: Starting and ending points are not warped.

LB_Keogh: Euclidean Distance of upper bound and lower bound from Sakoe-Chiba band.

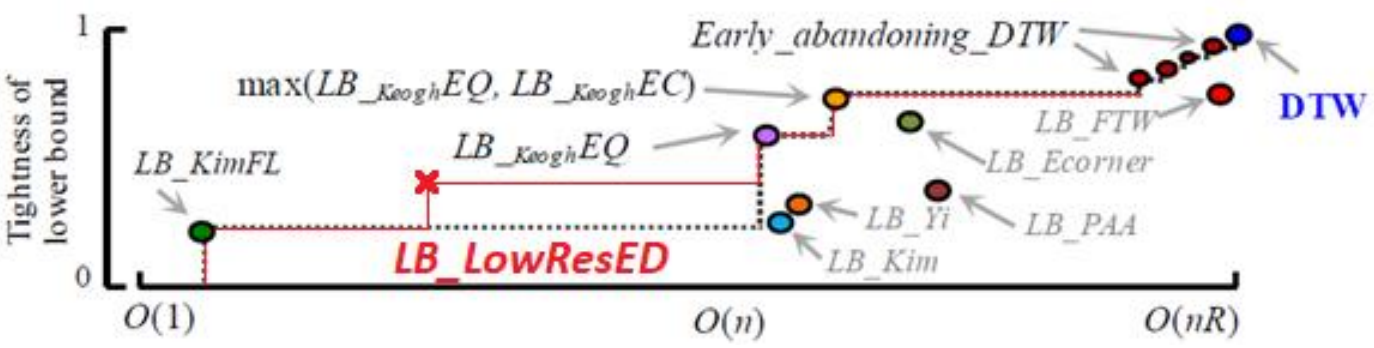
- Dynamic time warping is computed only if the lower bound is lower than distance threshold.



Source: Rakthanmanon, Thanawin, et al. "Searching and mining trillions of time series subsequences under dynamic time warping." Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2012.

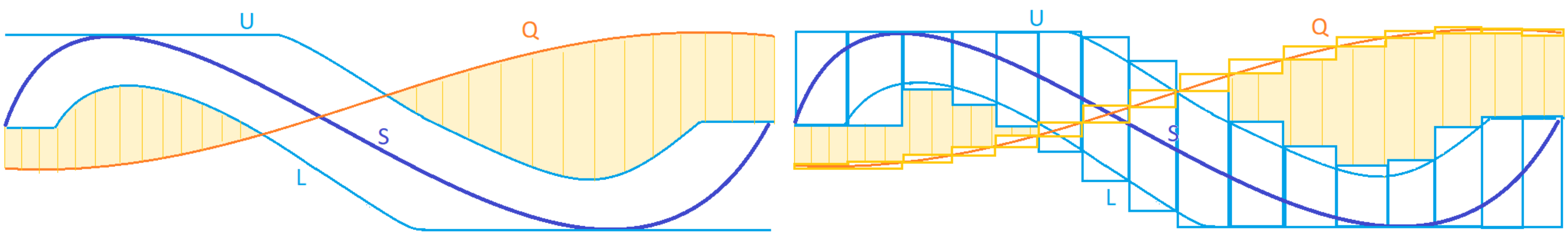
Proposed Method: *LB_LowResED*

- UCR Suite suggests that two cascading lower bounds, *LB_Kim* and *LB_Keogh* should be used.
- My proposal is to insert one lower bound function, namely *LB_LowResED*, to improve the pruning ability of lower bounds.
- It is based on low-resolution technique, which provides a lower bound for Euclidean distance (i.e. *LB_Keogh).*



Low Resolution Technique

- Given that the query length is *N*, we shall divide both query and subsequence to blocks of length = *k*.
- To compute *LB_LowResED*, we need to compute the Euclidean distance of (*N*/ *k*) blocks only.
- Low resolution technique is proposed by Eamonn Keogh. He designed a new lower bound function *LB_PAA* for **fixed-length** queries. Low-resolution block are indexed in R-tree.
- *LB_LowResED* is a modification of *LB_PAA,* which is applicable for **arbitrary-length** queries. It is designed for sequential search.



Left: *LB_Keogh* ; Right: Low bound of *LB_Keogh*, which is based on low-resolution technique

It is *LB_PAA* from the paper: Keogh, Eamonn, et al. "Supporting exact indexing of arbitrarily rotated shapes and periodic time series under Euclidean and warping distance measures." *The VLDB journal* 18.3 (2009): 611-630.

LB_LowResED algorithm

When answering the first query:

(i) Run unmodified UCR Suite.

(ii) Construct low-resolution sequence which is normalized in fixed-length.

When answering the next queries:

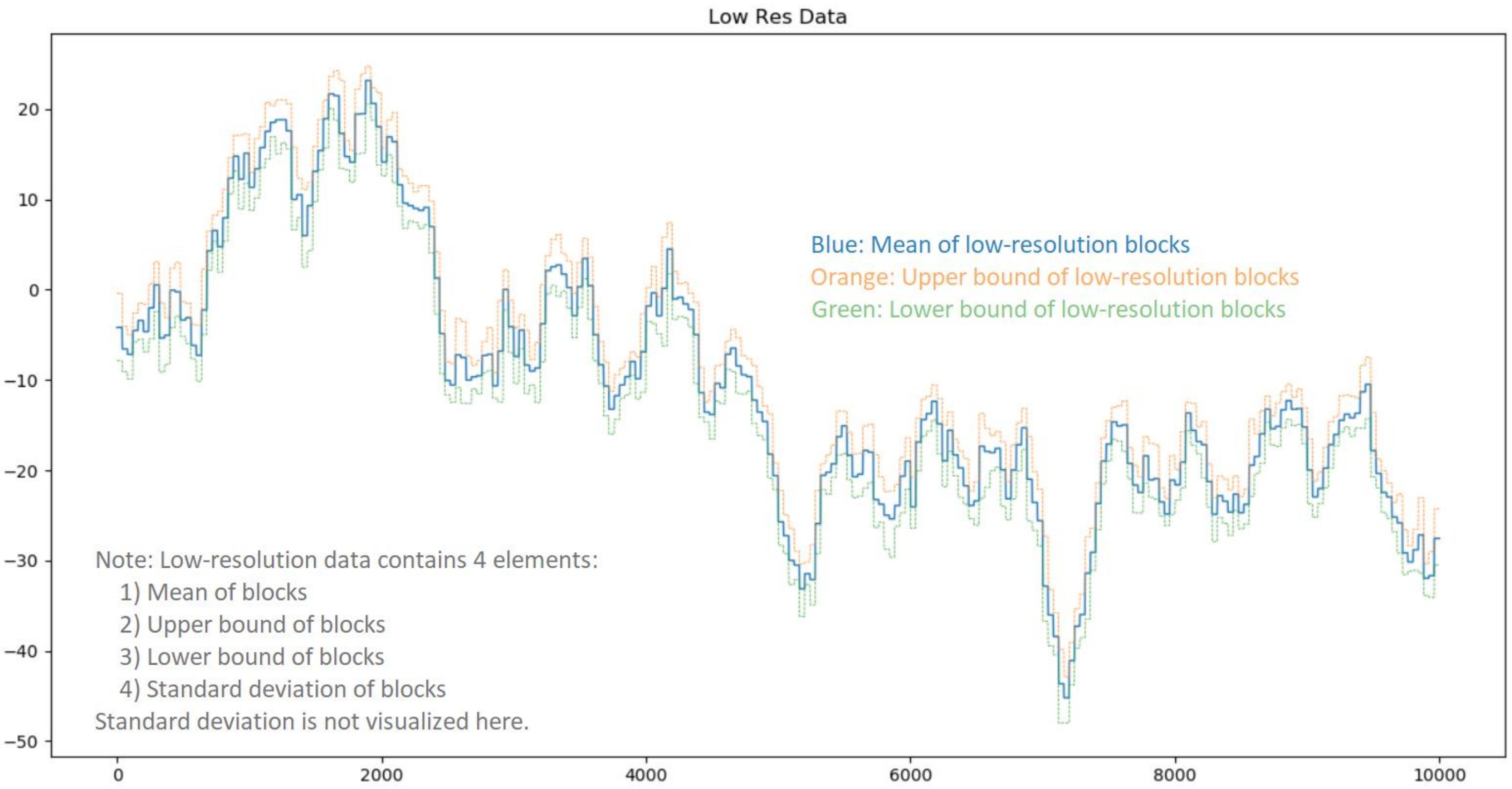
(i) Construct renormalized low-resolution subsequences. The renormalization depends on the length of query.

(ii) Compute distance between renormalized low-resolution subsequences and low-resolution query. Since both query and subsequences are in low-resolution form, the computation is fast. This distance is *LB_LowResED*.

(iii) If the lower bound distance is smaller than the best-so-far distance, we will test the subsequence with next cascading lower bound, i.e. *LB_Keogh* for dynamic time warping.

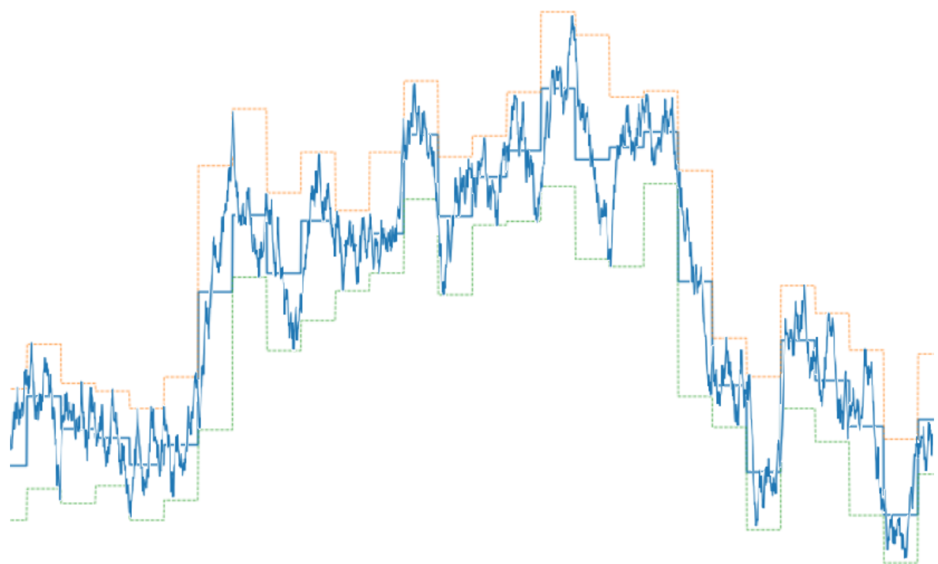
Further details would be explained in demonstration.

Algorithm: Construct Low-resolution Sequence



Algorithm: Construct Low-resolution Sequence

- To construct low-resolution sequence, the raw sequence is divided into blocks with a fixed-length.
- For each block, we store four parameters of that block :
  1) Mean,
  2) Standard deviation,
  3) Maxima, and
  4) Minima.

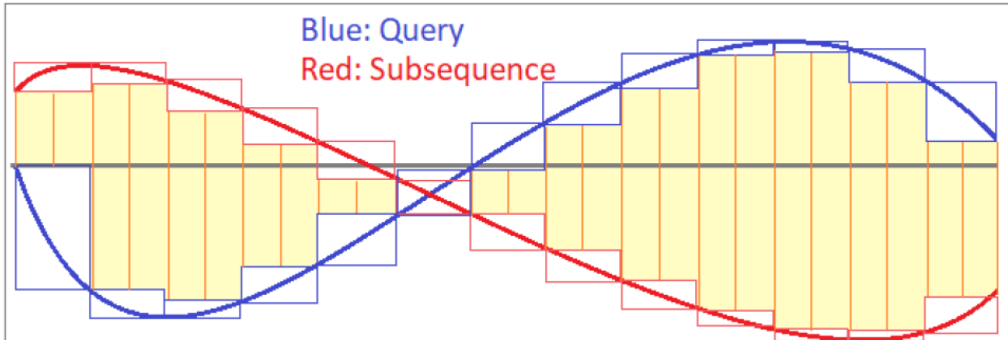


Algorithm : Construct Normalized Blocks

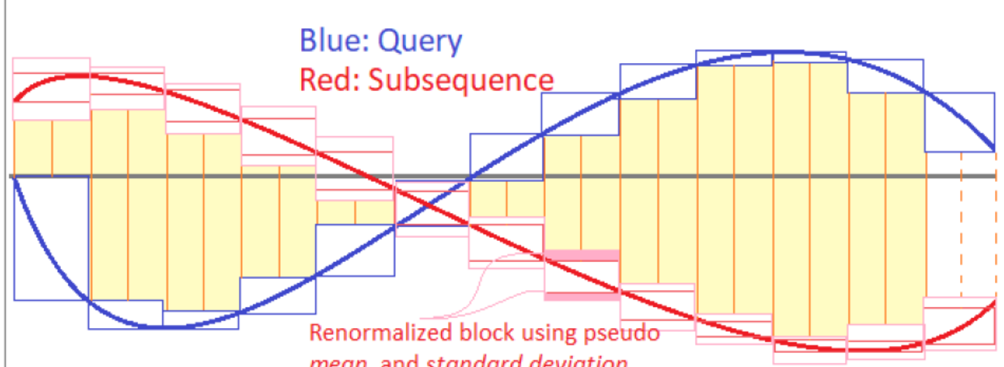
- Our goal is to compute distance between normalized query blocks and subsequence blocks.
- There are two challenges:
  1) How to normalize the low-resolution blocks subsequence, without knowing the actual mean and standard deviation?
  2) How to solve the shifting window problem for comparing the distance between normalized query and subsequence blocks?

Algorithm: Renormalization of Low-resolution Blocks

- Ideally, we should normalize a subsequence blocks using **actual *mean*** and **actual *standard deviation***.
- To accelerate the computing, we should not read the raw sequence.
- Hence, we cannot obtain the actual *mean* and *standard deviation* for normalization.
- We will obtain some **pseudo *mean*** and **pseudo *standard deviation*** for normalization.
- The pseudo *mean* and *standard deviation* are obtained from low-resolution sequence.



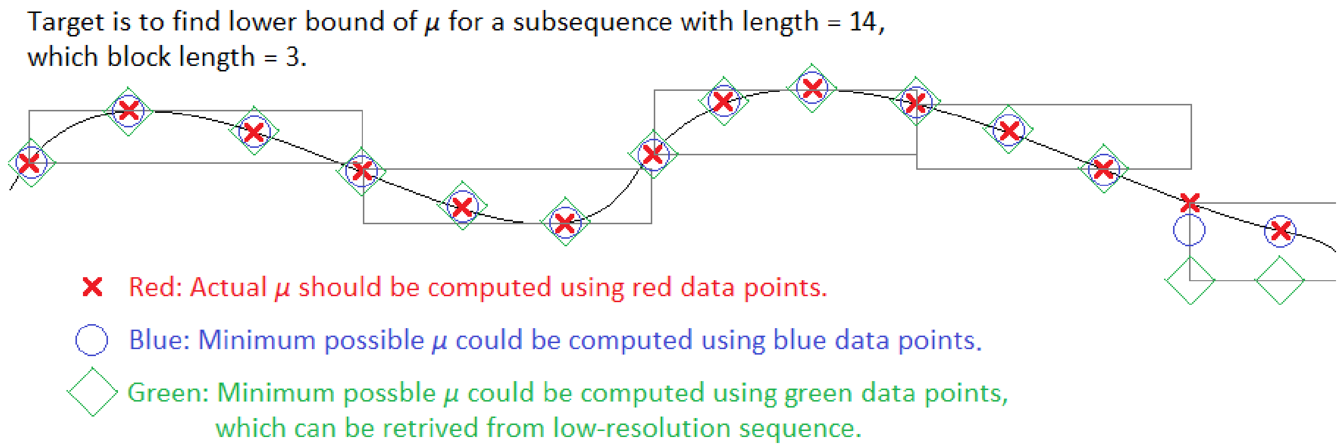
Ideal lower bound based on normalized MBR



Acceptable lower bound based on normalized MBR, which (1) use upper bound for maxima & lower bound of minima, and (2) truncated.

Renormalization of Low-resolution Blocks

- We can obtain maximum (or minimum) possible *mean* and *standard deviation* from low-resolution sequence.



- We could normalize the maxima (or minima) of low-resolution blocks using these maximum (or minimum) possible *mean* and *standard deviation.*
- Low-resolution normalized blocks are obtained.

Renormalization of Low-resolution Blocks

- Computation of maximum (or minimum) possible ***mean***:
- Details should be referred to the dissertation.

Note that $\mu_S = (s_1 + s_2 + \ldots + s_{a*k} + s_{a*k+1} + \ldots + s_{a*k+b}) / (a*k+b)$

$\geqq (s_1 + s_2 + \ldots + s_{a*k} + min(s_{a*k+1}, \ldots, s_{a*k+b}) * \mathrm{b}) / (a*k+b)$

$\geqq (s_1 + s_2 + \ldots + s_{a*k} + min(s_{a*k+1}, \ldots, s_{(a+1)*k}) * \mathrm{b}) / (a*k+b)$

We define $\mu_S^{min} = (s_o + s_1 + s_2 + \ldots + s_{a*k} + min(s_{a*k+1}, \ldots, s_{(a+1)*k}) * \mathrm{b}) / (a*k+b)$, which can be computed using low-resolution sequence:

- $(s_1 + s_2 + \ldots + s_{a*k})$ can be obtained because we have mean of every block.
- $min(s_{a*k+1}, \ldots, s_{(a+1)*k})$ can be obtained because we have minima of next block.
- $(a*k+b)$ is length of query, which is given.

We define $\mu_S^{max} = (s_o + s_1 + s_2 + \ldots + s_{a*k} + max(s_{a*k+1}, \ldots, s_{(a+1)*k}) * \mathrm{b}) / (a*k+b)$, which can be computed using low-resolution sequence too.

Renormalization of Low-resolution Blocks

- Computation of minimum possible ***standard deviation***:
- Details should be referred to the dissertation.

Note that $\sigma_S^2 = (s_1^2 + s_2^2 + \ldots + s_{a*k}^2 + s_{a*k+1}^2 + \ldots + s_{a*k+b}^2) / (a*k+b) - \mu_S^2$

$\geq (s_1^2 + s_2^2 + \ldots + s_{a*k}^2 + s_{a*k+1}^2 + \ldots + s_{a*k+b}^2) / (a*k+b) - max(\mu_S^2)$,

which $max(\mu_S^2) = (\mu_S^{max})^2$ for positive μ_S, and

$max(\mu_S^2) = (\mu_S^{min})^2$ for negative μ_S.

We obtained μ_S^{min} and μ_S^{max} by algorithm for minimum/ maximum possible μ_S.

If both μ_S^{min} and μ_S^{max} are positive, then μ_S is positive.

If both μ_S^{min} and μ_S^{max} are negative, then μ_S is negative.

If μ_S^{min} is negative and μ_S^{max} is positive, we choose $max(\mu_S^2) = \max((\mu_S^{min})^2, (\mu_S^{max})^2)$

$\geq (s_1^2 + s_2^2 + \ldots + s_{a*k}^2 + min(s_{a*k+1}^2, \ldots, s_{a*k+b}^2)*\mathrm{b})/(a*k+b) - max(\mu_S^2)$

$\geq (s_1^2 + s_2^2 + \ldots + s_{a*k}^2 + min(s_{a*k+1}^2, \ldots, s_{(a+1)*k}^2)*\mathrm{b})/(a*k+b) - max(\mu_S^2)$,

which $min(s_{a*k+1}^2, \ldots, s_{(a+1)*k}^2)$ is square of the data point that

If both minima and maxima of next block is negative, the data point chosen is maxima of next block.
If both minima and maxima of next block is positive, the data point chosen is minima of next block.
If the minima and maxima of next block is one positive and one negative, the data point chosen is zero.

We define $\sigma_S^{min} = [(s_1^2 + s_2^2 + \ldots + s_{a*k}^2 + min(s_{a*k+1}^2, \ldots, s_{(a+1)*k}^2)*\mathrm{b})/(a*k+b) - max(\mu_S^2)]^{1/2}$, which can be computed using low-resolution sequence.

Renormalization of Low-resolution Blocks

- Computation of maximum possible ***standard deviation***:
- Details should be referred to the dissertation.

Note that $\sigma_S^2 = (s_1^2 + s_2^2 + \ldots + s_{a*k}^2 + s_{a*k+1}^2 + \ldots + s_{a*k+b}^2) / (a*k+b) - \mu_S^2$

$\leqq (s_1^2 + s_2^2 + \ldots + s_{a*k}^2 + s_{a*k+1}^2 + \ldots + s_{a*k+b}^2) / (a*k+b) - min(\mu_S^2)$,

which $min(\mu_S^2) = (\mu_S^{min})^2$ for positive μ_S, and

$min(\mu_S^2) = (\mu_S^{max})^2$ for negative μ_S.

We obtained μ_S^{min} and μ_S^{max} by algorithm for minimum/ maximum possible μ_S.

If both μ_S^{min} and μ_S^{max} are positive, then μ_S is positive.

If both μ_S^{min} and μ_S^{max} are negative, then μ_S is negative.

If μ_S^{min} is negative and μ_S^{max} is positive, we choose $min(\mu_S^2) = 0$.

$\leqq (s_1^2 + s_2^2 + \ldots + s_{a*k}^2 + max(s_{a*k+1}^2, \ldots, s_{a*k+b}^2)*\mathrm{b})/(a*k+b) - min(\mu_S^2)$

$\leqq (s_1^2 + s_2^2 + \ldots + s_{a*k}^2 + max(s_{a*k+1}^2, \ldots, s_{(a+1)*k}^2)*\mathrm{b})/(a*k+b) - min(\mu_S^2)$,

which $max(s_{a*k+1}^2, \ldots, s_{(a+1)*k}^2)$ is square of the data point that

If both minima and maxima of next block is negative, the data point chosen is minima of next block.

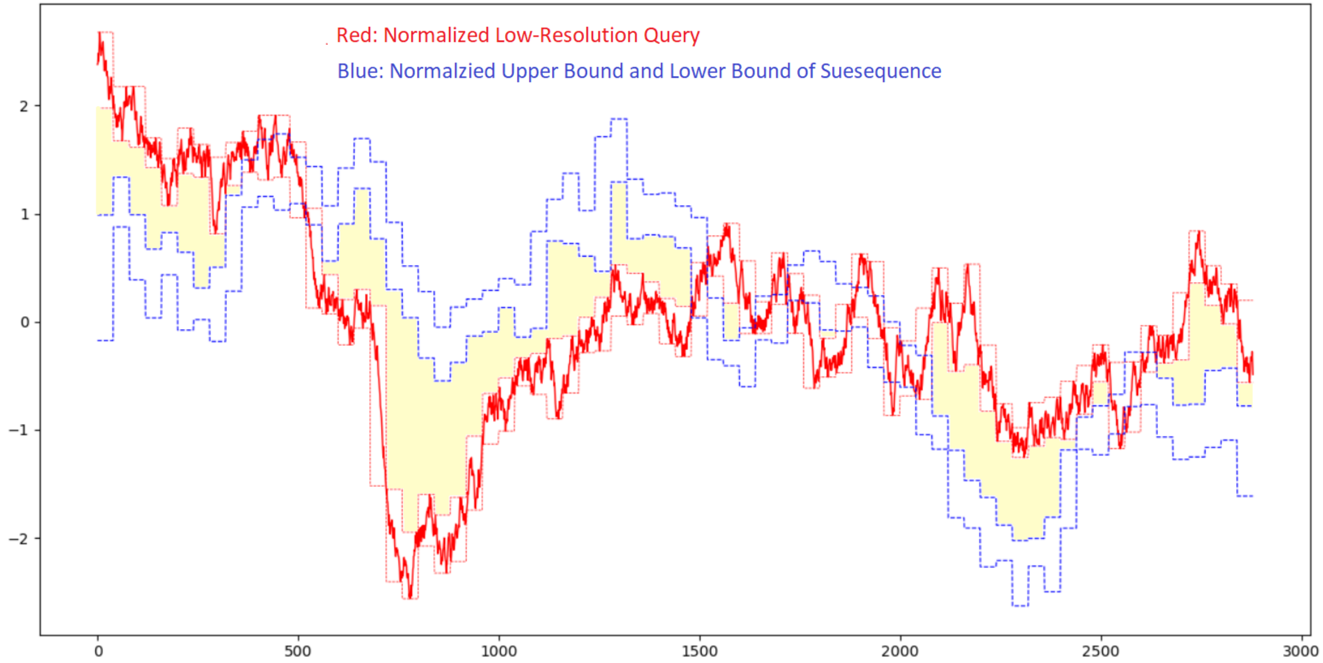
If both minima and maxima of next block is positive, the data point chosen is maxima of next block.

If the minima and maxima of next block is one positive and one negative, the data point chosen is $max($ abs(maxima of next block), abs(minima of next block)).

We define $\sigma_S^{max} = [(s_1^2 + s_2^2 + \ldots + s_{a*k}^2 + max(s_{a*k+1}^2, \ldots, s_{(a+1)*k}^2)*\mathrm{b})/(a*k+b) - min(\mu_S^2)]^{1/2}$, which can be computed using low-resolution sequence.

Shifting Window

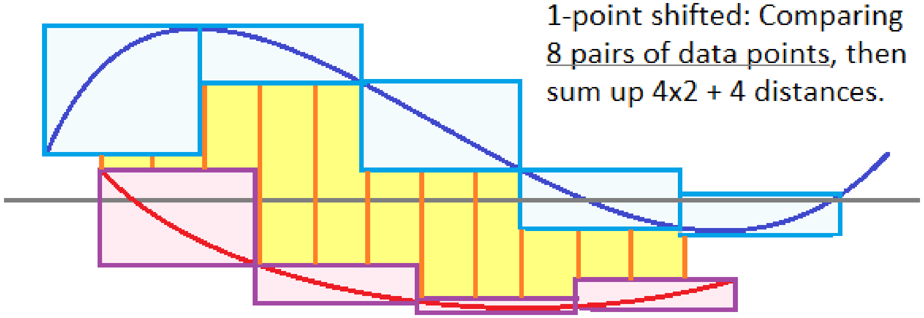
- In *LB_LowResED*, we compute the distance between low-resolution blocks.

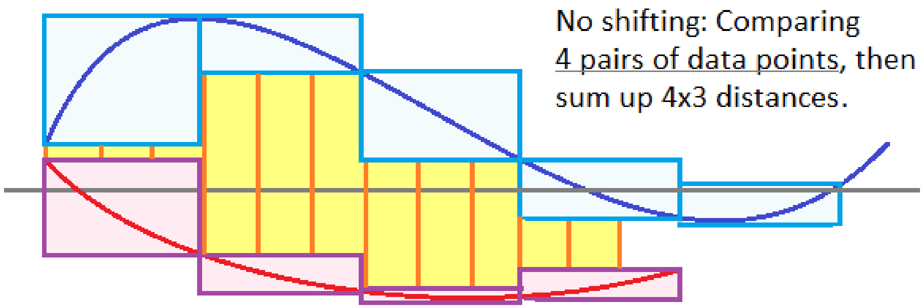


- How to compute the distance for next subsequence?

Shifting Window

- Low-resolution comparison converted the problem from (a*k + b) comparison to a comparison if b = 0, i.e. no shifting window problem.
- For b > 0, the number of comparisons is would be greater than a.

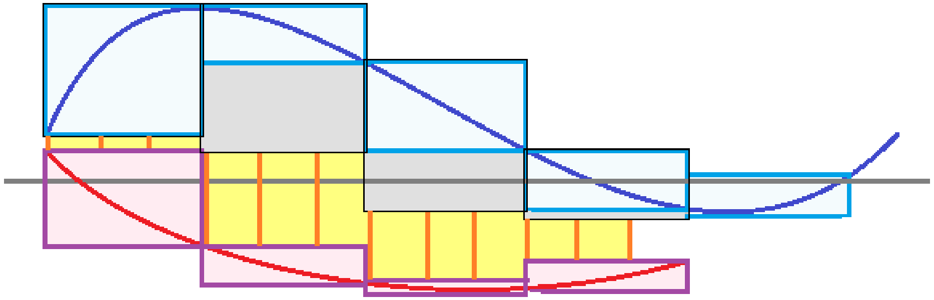




Red: Normalized query blocks
Blue: Normalized dataset blocks

Shifting Window

- Shifting window problem can be solved by merging low-resolution blocks.
- We compare one query block with one subsequence block and the next block.
- Effectively, we compare query with the next *k* shifted subsequences at the same time.



Solution of shifted window: effectively compare query with corresponding subsequence block and next block. Still it is comparing 4 pairs of data points, then sum up 4x3 distances.

Experiment – Various Datasets

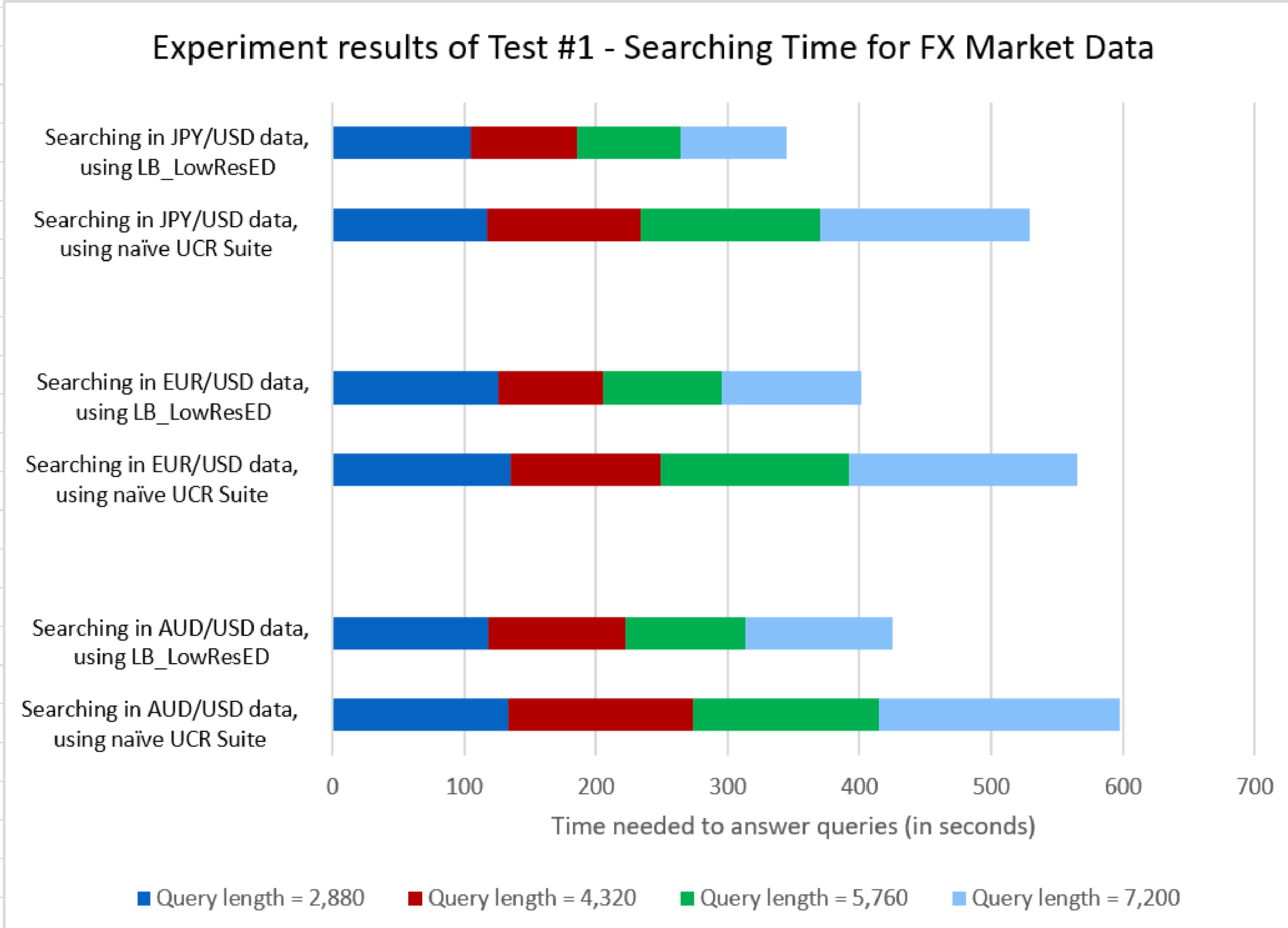
Test environment:

CPU: Intel Core i5-7300HQ 2.5 GHz. Single core is used.

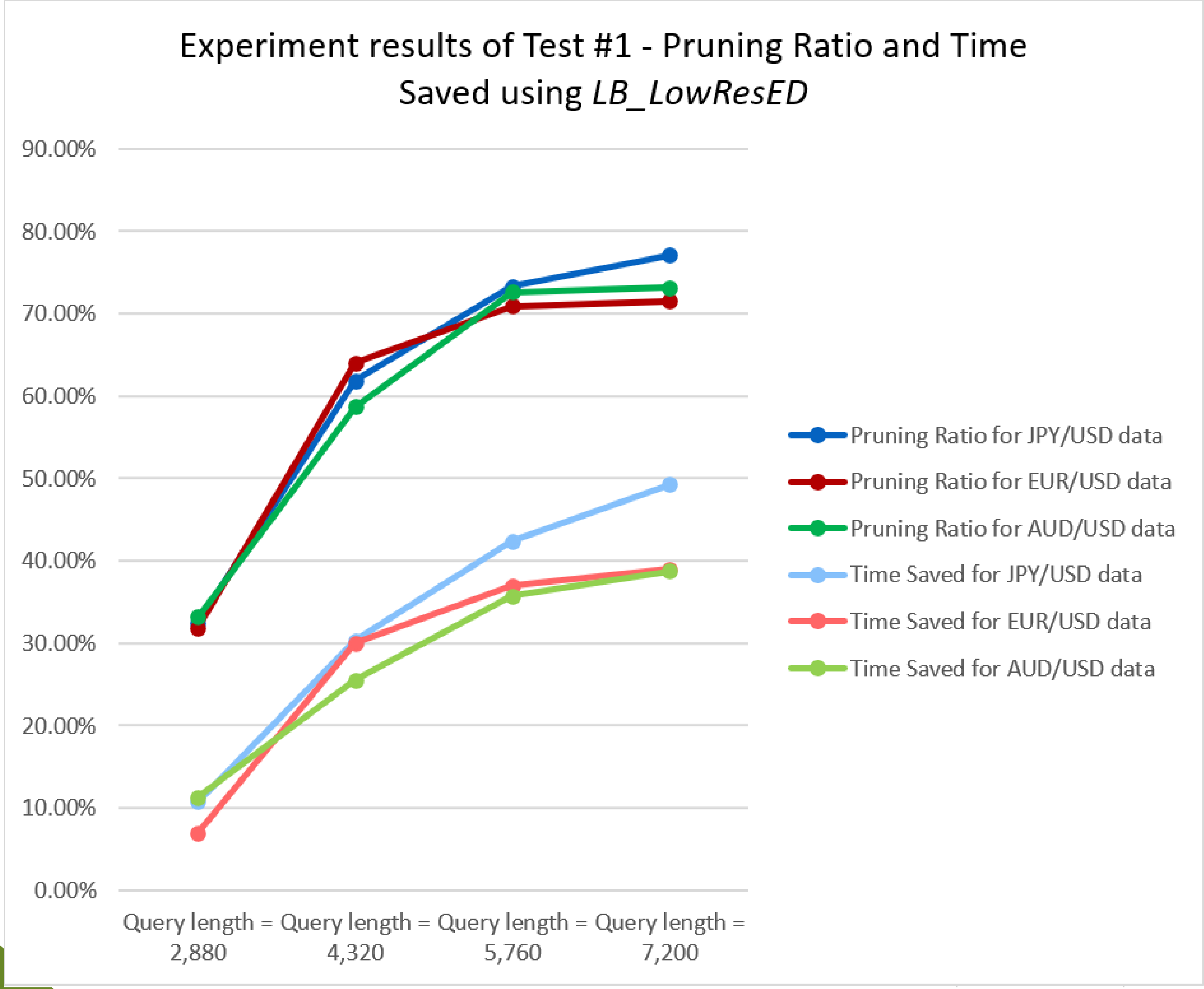
Software: The script is run in Python 3.6.6.

- We perform experiment for search in FX Market data.
- Queries are random walk generated from program. Total 5 queries.
- Sequence is price of foreign currency per minute for 3 year, obtained from HistData.com. Dataset of 3 year has about 1 million data points. Block length is 40.
- We measure the Euclidean distance between query and subsequence. Top 3 nearest neighbours are filtered out.
- We ask the same query thrice. The average of times needed for answering the query are recorded and compared.

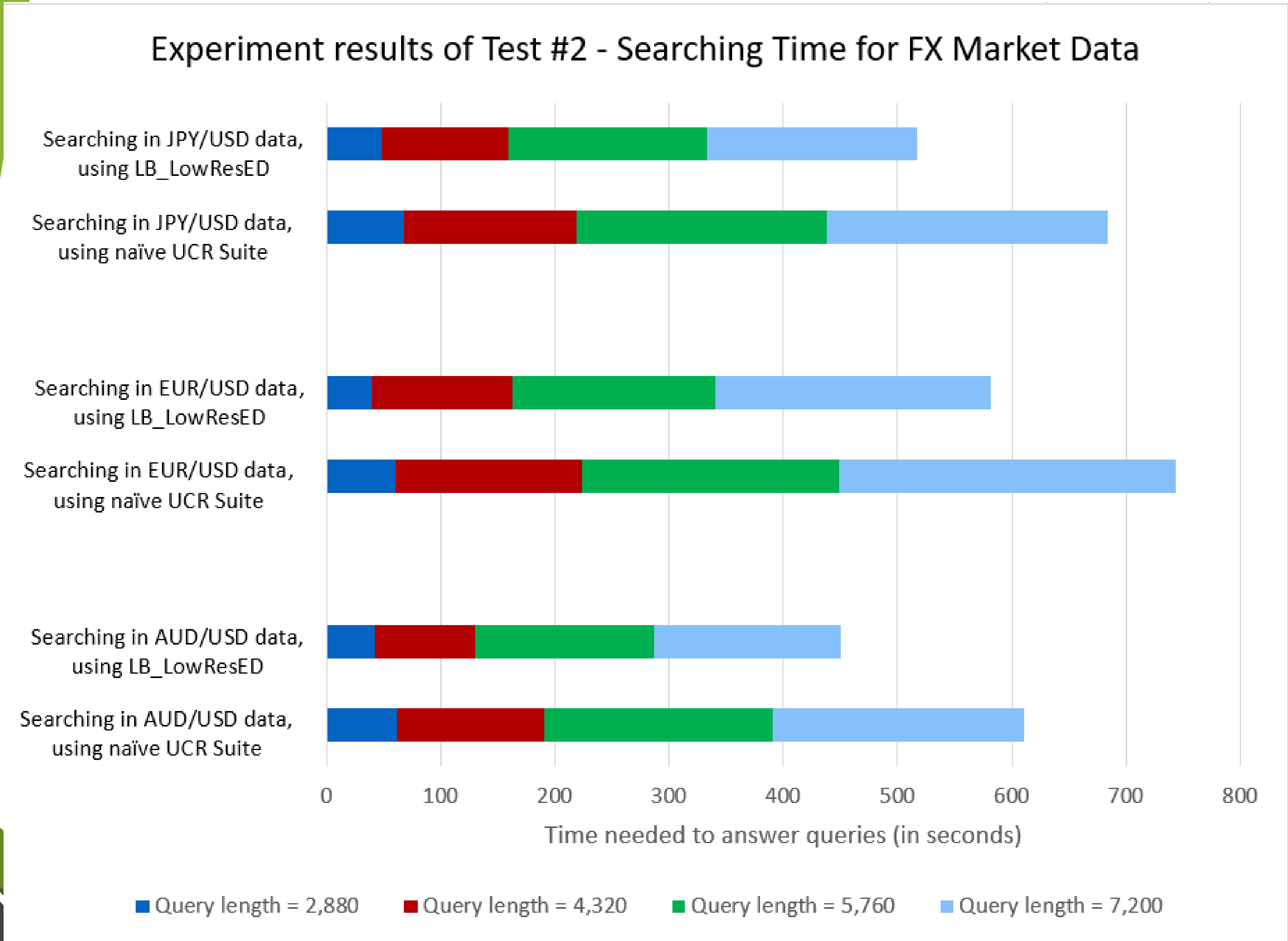
Experiment – Query #1



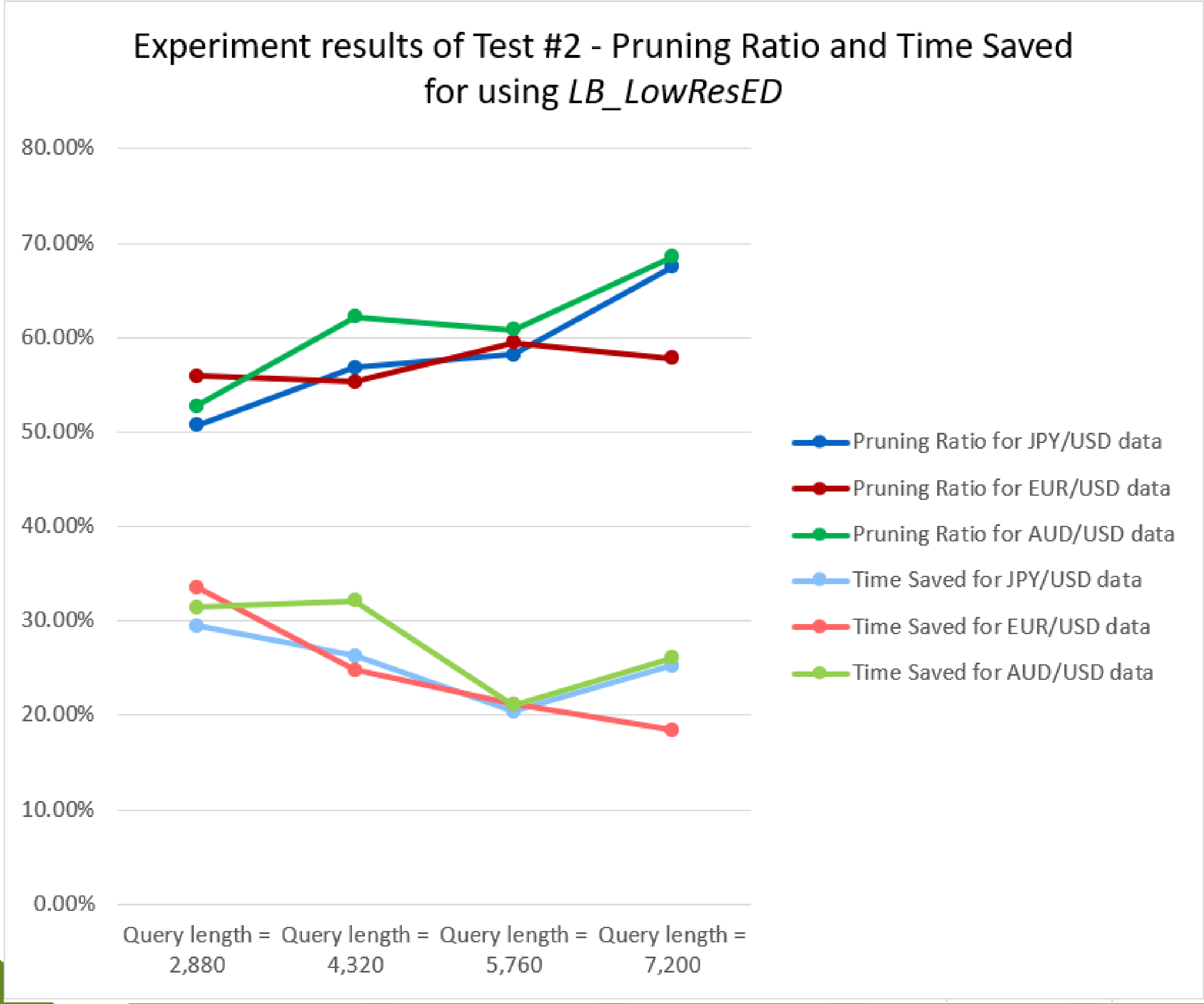
Experiment – Query #1



Experiment – Query #2



Experiment – Query #2



Experiment – Various Block Lengths

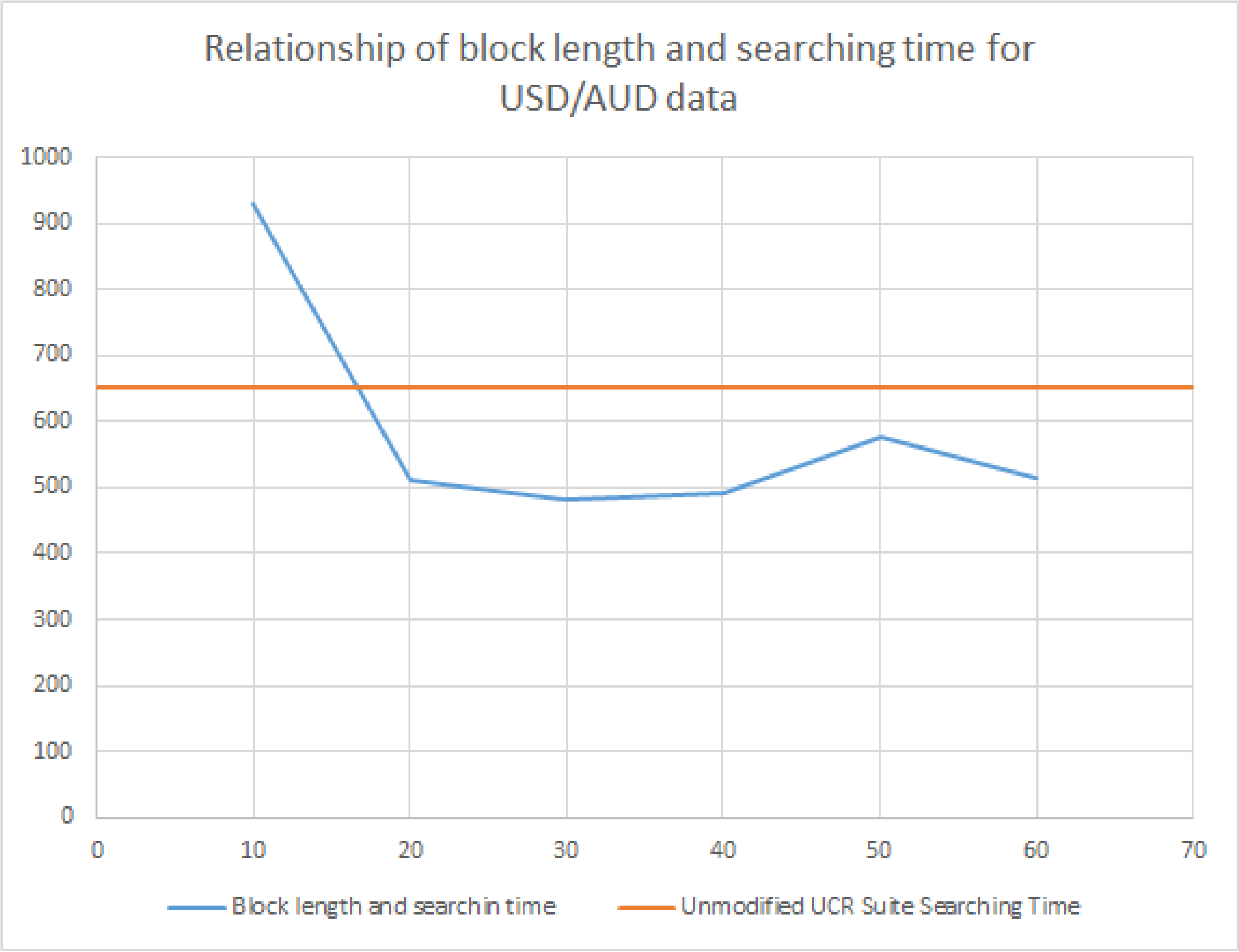
Test environment:

CPU: Intel Core i5-7300HQ 2.5 GHz. Single core is used.

Software: The script is run in Python 3.6.6.

- We perform experiment for search in USD/AUD FX data.
- Queries are random walk generated from program. Total 5 queries.
- Sequence is price of foreign currency per minute for 3 year, obtained from HistData.com. Dataset of 3 year has about 1 million data points. Block length is 40.
- We measure the Euclidean distance between query and subsequence. Top 3 nearest neighbours are filtered out.
- We ask the same set of queries for different block lengths. The average of times needed for answering the queries are recorded and compared.

Experiment – Various Block Lengths



Conclusion

- The state-of-art solution for similarity search for time series subsequence under dynamic time warping is UCR Suite.
- UCR Suite suggests that lower bounds of dynamic time warping could be *LB_Keogh* and *LB_Kim*, which are measured in Euclidean distance.
- The new lower bound function, *LB_LowResED*, is designed for accelerating sequential search of time series data if similarity is measured in Euclidean distance.
- It is based on low-resolution technique.
- If a suitable block length is chosen, generally it can save 20% to 30% of searching time.

Future Work

- We improved UCR Suite by modifying the cascading lower bound technique.
- For future works, we could attempt to improve UCR Suite by modifying early abandon and reordering techniques.
- One direction is to study the relation between shape of query, block length of *LB_LowResED* and the effectiveness of acceleration.