



UNIVERSITÉ
PARIS
DESCARTES



Université de Paris

QITONG WANG and THEMIS PALPANAS

University of Paris, France

Summarization Project

By: *BOUDEGZDAME Nada Ikram*

KACIMI Imad Oualid

DJELLAL Mohamed Aniss

Keywords:

Time Series, Time-related database, Similarity Search, Summarization, Discrete Wavelet Transform

I. Introduction:

Large amount of data is being produced all over the world. The inventions of social networking websites and introduction of IT equipment in a large number of fields are continuously adding to the data repositories all over the world. Most of this data is time dependent meaning time series. Time Series (TS) are an ordered sequence of values of a variable at equally spaced time intervals.[8] It can be taken on any variable that changes over time. Meaning that data during in series of particular time periods or intervals. Time series analysis is a statistical technique that deals with this kind of data. It includes methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data. [9] Recently there has been much interest in the problem of similarity search in time series databases. Since it's an important aspect of knowledge discovery in time series. This is hardly surprising given that time series account for much of the data stored in business, medical and scientific databases.

However, storing this huge data needs lot of memory moreover analysis of this data for extracting these features from it consumes lot of time. Thus, to cut back the time required for extracting these information the given time series must be represented in lower dimension thus storing only necessary values that depicts some reasonable information and helps in important data mining tasks.

In this report, we present our solution to solve the problem of similarity search in databases using summarization as a way to fasten this process. An interpretation to the result of our solution will be presented in further suction.

II. Problem Formulation:

Similarity search is the principle of searching spaces of objects where the only available comparator is the similarity between any pair of objects. Time series databases are often extremely large. So, the ability to search through the database becomes unreasonably time consuming. In addition the growing number of times series databases makes time crucial, especially given the magnitude of many time series databases since they account for much of the data stored in business, medical and scientific databases.

The key to a good and a fast similarity search solution, is to summarize data. Since summarization is the process of shortening a set of data, to create a subset or a summary that represents the most important or relevant information within the original content. It leads to a less memory use to represent the original dataset. In addition, summarization is a key data mining concept which involves techniques for finding a compact description of a dataset. The purpose of this project is to summarize times series in order to bring space efficiency, and use the result to solve the similarity search problem.

Given the importance of many time series databases, much research has been dedicated to speeding up the search

process. Therefore, many solutions exist for both summarization problem and similarity search problem.

Frist let's start with solution for summarizing data:

- **Discrete Wavelet Transform (DWT)**

It's an implementation of the wavelet transform using a discrete set of the wavelet scales and translations obeying some defined rules. In other words, this transform decomposes the signal into mutually orthogonal set of wavelets, which is the main difference from the continuous wavelet transform (CWT).[4]

Using DWT, a time series is represented in terms of a finite length, fast decaying, oscillating, and discretely sampled wave form (mother wavelet), which is scaled and translated in order to create an orthonormal wavelet basis. Each function in the wavelet basis is related to a real coefficient; the original series is reconstructed by computing the weighted sum of all the functions in the basis, using the corresponding coefficient as weight.

- **Swinging Door (SD)**

SD is a data compression technique that belongs to the family of piecewise linear trending functions. SD has been compared to wavelet compression. The SD algorithm employs a heuristic to decide whether a value is to be stored within the segment being grown or it is to be the beginning of a new segment. Given a pivot point, which indicates the beginning of a segment, two lines (the "doors") are drawn from it to envelop all the points up to the next one to be considered. The envelop has the form of a triangle according to a parameter that specifies the initial amplitude of the lines. The setup of this parameter has impact on the data compression level.

- **Piecewise Aggregate Approximation (PAA)**

It's widely used in time series data mining because it allows to discretize, to reduce the length of time series. However, it requires setting one parameter: the number of segments to consider during the discretization. The optimal parameter value is highly data dependent in particular on large time series. This method represents a time series by computing average of sequential fixed length segments. It transforms a time series of n points in a new one composed by p segments (with $p > n$), each of which is of size equal to n/p and is represented by the mean value of the data points falling within the segment. [5]

PAA leads directly to another representation called Symbolic Aggregate Approximation, shortened to SAX. This takes the output of PAA and assigns a string representation to the graph. By providing a string representation, the overall data that is required to be stored is less than other data mining

methods such as Discrete Fourier Transform (DFT) and Wavelet Transformation.

As for the similarity search problem, we present the following one:

- **Dynamic Time Warping (DTW):**

DTW is a distance-like measure that allows elastic shifting of the time axis, which has shown good empirical performance for time series classification. Discord Detection methods apply Euclidean distances or DTW between subsequences for anomaly detection, while subsequence clustering methods find clusters of similar subsequences.

This technique is currently the most relevant similarity measure between sequences for a large panel of applications, since it makes it possible to capture temporal distortions.

III. Why using DWT?

Discrete Wavelet Transform (DWT) is a mathematical origin technique. It's highly appropriate for noise filtering, data reduction, and singularity detection, which makes it a good choice for time series data processing. This technique has been used extensively in a wide range of areas, such as in signal processing, and specifically it is frequently employed for research in signal compression, image enhancement and noise reduction. [3]

Time series data analysis and mining is another area where researchers have recently applied DWT techniques due to its favorable properties:

- DWT is a very useful technique for time series data processing of many aspects such as data dimensionality reduction, noise reduction, and multiresolution analysis.[3]
- DWT can create separate time series from the original time series. The original information will be distributed into these different time series in the form of wavelet coefficients. Therefore, DWT is considered an orthonormal transformation, meaning it allows reconstruction and preserves the original information [3].

The main reason for choosing the DWT is the orthonormal property that will help us reduce the amount of information that will be lost.

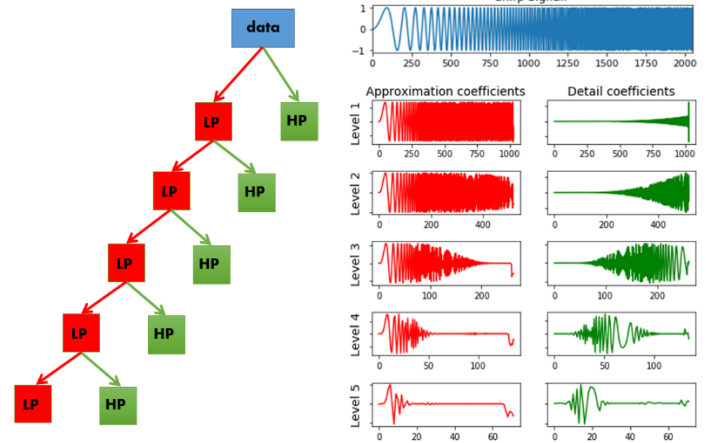
IV. Description of the solution:

General Idea:

This technique is mainly divided into two phases, first data reduction and reducing the details coefficients. As mentioned before, for the task of data reduction we have chosen to use the discrete wavelet transform using pywavelet in python that represent a library for manipulating wavelets.

Applying discrete wavelet transform on time-series data will divide the original series into two set of coefficients:

- Set of approximation coefficient: represents the reduced data.
- set of detail coefficient: represents the coefficients that are needed in the reconstruction of the original data.



By applying the DWT again on the approximation coefficients of the previous DWT, we get the wavelet transform of the next level as shown in the previous picture. We have to mention that, at each next level, the size of the data is sampled down by a factor of 2.

In our case we want to produce three newest data set of size 128, 64 and 32 bytes. In order to do it we have to apply DWT multiple times, respectively 3, 4 and 5 level of transformation to the original data of size 1024 byte, then saving the approximation coefficient from the last level of transformation.

1. Description of data reduction:

• Data Reduction:

As shown in figure 2, the data reduction function is given. It takes as input both original data series and the level of DWT that will be applied. The output of this function is the data returned from the last DWT and a list containing the detail coefficients of each transformation level.

We have to mention that the list of details coefficient contains multiple list, the number of those list depend on how many times we apply the DWT, we will have:

- A list of 3 lists for the reduction to 128 bytes.
- A list of 4 lists for the reduction to 64 bytes.
- A list of 5 lists for the reduction to 32 bytes.

```

1
2
3
4 def time_series_reduction(timeSeries,level):
5     data=timeSeries;
6     coeffs=[];
7     i=0;
8     while(i<level){
9         data,coef=pywt.dwt(data,'sym5',mode="per");
10        coeffs.append(coef);
11    }
12    return data,coeffs;

```

Figure 2 - Data Reduction Function.

- **Detail Coefficients Storing:**

As explained before, the results of summarizing the time series are the data reduced and a coefficient array. Both of the results are important for the reconstruction of the data, which means both of them must be stored. However, the size of the coefficient array is equal to the size of the original data. Meaning that storing the array as it won't bring space efficiency. So, the coefficient needs to be summarized as well. The technique applied consists of saving the coefficient with the largest absolute value. Different combination for the summary were tested and the one with the least error was chosen.

First of all, for the time series summarized into 128 bytes. The result of the summarization gave three coefficient table, one with a length of 128, and the second with a length of 64, and a third with a length of 32. Which make a total size of $128 * 4 \text{ byte/float} + 64 * 4 \text{ byte/float} + 32 * 4 \text{ byte/float} = 896$ bytes. The table below represents the different possibilities tested.

Final length of first table	Final length of second table	Final length of third table	The error after re-construction
8	8	8	1.81
8	16	8	3.04
16	8	8	3.45

The first method was chosen since it gives the least error. The final size was $16 * 4 \text{ byte/float} + 8 * 4 \text{ byte/float} + 8 * 4 \text{ byte/float} = 128$ bytes.

Secondly, for the time series summarized into 64 bytes, the result of the summarization gave four coefficient table, one with a length of 128, and the second with a length of 64, the third with a length of 32, and the fourth with a length of 16 floats. The table below represents the different possibilities tested.

Final length of first table	Final length of second table	Final length of third table	Final length of fourth table	Error after re-construction
4	4	4	4	2.83
8	4	2	2	3.02
4	8	2	2	4.23
2	4	8	2	5.63
2	2	4	8	6.65

The first method was chosen since it gives the least error. The final size was $4 * 4 \text{ byte/float} + 4 * 4 \text{ byte/float} + 4 * 4 \text{ byte/float} + 4 * 4 \text{ byte/float} = 64$ bytes.

Finally, for the time series summarized into 32 bytes, the result of the summarization gave four coefficient table, one

with a length of 128, and the second with a length of 64, the third with a length of 32, the fourth with a length of 16, and the fifth with a length of 8 floats. The number of possibilities is large, we tested only the combination that seemed promising to us looking at the results of the previous summarization. The table below represents the different possibilities tested.

Final length of first table	Final length of second table	Final length of third table	Final length of fourth table	Final length of fifth table	Error after re-construction
1	1	2	2	2	9.81
1	2	1	2	2	8.03
1	2	2	1	2	8.65
1	2	2	2	1	7.44
2	1	2	2	1	7.04
2	2	1	1	2	5.23
2	2	2	1	1	3.87

The last method was chosen since it gives the least error. The final size was $2 * 4 \text{ byte/float} + 2 * 4 \text{ byte/float} + 2 * 4 \text{ byte/float} + 1 * 4 \text{ byte/float} + 1 * 4 \text{ byte/float} = 32$ bytes.

- **Merging Position with Detail Coefficients:**

We have noticed that our detail coefficient values are between -9 and 9, so we had the idea of merging our coefficients positions with the detail coefficients values, to save more space, as we can only save the same amount of approximation coefficients and details coefficients.

Let us take the example a detail coefficient value of 1.89798 that have the position 15 the merging result will give us: 151.89798.

```
def merge_coeff_indice(indice,coeff):
    merged_indice_coeff=[];
    i=0;
    while(i < length(coeff)){
        val = indice[i]*10+abs(coeff[i])
        if (coeff[i] < 0)
            val*=-1
        merged_indice_coeff.append(val)
        i++;
    }
    return merged_indice_coeff
```

Figure 3 - Merging Detail and Coefficient Positions Function.

- **Dataset Reduction:**

After defining the functions needed to reduce the data. We apply them on all the time series in the dataset (both synthetic or seismic original dataset).

We define the function bellow, taking as input the path of the dataset and reduce it according to the level of the DWT transformation that would be applied to each time series. The output of this function is two data frames one containing the reduced dataset (128,64 or 32 byte), and the other containing the positions and values of detail coefficients.

```

i = 0;
while ( i < 50 000) {
- Application of the reduction of time series and providing the
reduced data and the coefficient;
}
-Saving the reduced data;
i = 0;
while ( i < 50 000) {
- Selection the relevant coefficient provided by the previous loop;
- Merging the position with the values of the relevant coefficient;
}
- Saving the coefficient reduced and merged;

```

Figure 4 - Dataset Reduction.

The result of reducing our original dataset (synthetic and seismic) of size 1024 byte into 128, 64 and 32 bytes will be saved as shown in the figure 5:





























	seismic_size50k_len256_znor...		_dm aniss	15 mars 2020 _dm an
	red_seismic_50k_128.bin		_dm aniss	1 avr. 2020 _dm anis
	red_seismic_50k_64.bin		_dm aniss	1 avr. 2020 _dm anis
	red_seismic_50k_32.bin		_dm aniss	1 avr. 2020 _dm anis
	coef_seismic_50k_128.bin		_dm aniss	1 avr. 2020 _dm anis
	coef_seismic_50k_64.bin		_dm aniss	1 avr. 2020 _dm anis
	coef_seismic_50k_32.bin		_dm aniss	1 avr. 2020 _dm anis
	synthetic_size50k_len256_zn...		_dm aniss	15 mars
	red_synthetic_50k_128.bin		_dm aniss	1 avr. 20
	red_synthetic_50k_64.bin		_dm aniss	1 avr. 20
	red_synthetic_50k_32.bin		_dm aniss	1 avr. 20
	coef_synthetic_50k_128.bin		_dm aniss	1 avr. 20
	coef_synthetic_50k_64.bin		_dm aniss	1 avr. 20
	coef_synthetic_50k_32.bin		_dm aniss	1 avr. 20

Figure 5- Reduction Output.

2. Description of data reconstruction:

General idea:

In order to reconstruct data both reduced data and detail coefficient are needed. The only coefficients needed for a perfect reconstruction are the approximation coefficients from the last level of transformation and the detail coefficients from every level of transformation. In our case we have stored both reduced data and some of the relevant detail coefficients

as explained above. The process of reconstructing data has two main tasks. First one is to reconstruct the detail coefficient and use IDWT function to reconstruct the original data.

Description of the solution:

• Detail coefficients reconstruction:

First of all, in the reconstruction of the coefficient we have to extract the positions and values of detail coefficient that have been stored in the reduction task. In order to do it, we provide a function that take the merged dataset stored previously, and return two datasets (coefficients values and positions).

Secondly, we produce a list containing multiple list of detail coefficients according to the storing technique chosen. Producing this list depends on the size of the reduced data as explained in the first reduction part.

After extracting the position of the coefficient, each coefficient will be placed at its position. As for the coefficient that didn't make it through the summarization, they will be replaced by a default value equal to zero. This value will not make neither a bad nor a good impact on the reconstruction which means it won't falsify the results. The result will be used by the IDWT function to reconstruct the original data.

• Time series reconstruction:

After reconstructing the details coefficient, the data can finally be reconstructed using the reduced data as well. The function bellow aims to reconstruct the original time series from a reduced one and its coefficient after reconstructing it as explained.

```

def data_reconstruction(data_reduced,coefficient):
    for i in range(len(coefficient)-1,-1,-1){
        data_reduced = pywt.idwt(data_reduced, coefficient[i]);
    }
    data_reconstructed=data_reduced
    return data_reconstructed

```

Figure 6- Original Time series Reconstruction.

• Average Error:

We provide a function that will calculate the square mean error between each reconstructed time series stored and original one. Each error will be stored in a list then we provide the average error of this list.

3. Description of Similarity Search:

By similarity search, we mean that given a query time series, the most similar time series in our dataset is retrieved. Time series are compared by calculating the sum of the distance between each point of the two-time series. The method used to calculate the distance between time series is the Euclidean distance. But calculating the distance between time series coded in 256bits is time consuming, so the summarized ones is used and then calculate their distance to make the search faster.

For each query, the search approach that we followed is the following:


```

closest = infinity
foreach timeseries:
    if closest > distance(summarized(timeseries),
summarized(query) ):
        d = distance (times series, query)
        if closest > d:
            closest = d

```

Since NumPy is used for efficient manipulations and operations on High-level mathematical functions, Multi-dimensional arrays, Linear algebra etc. It was used to calculate the Euclidian distance in a faster way. For that we define the next method.

```

def euclidianDistance(X,Y):
    return np.linalg.norm(X.values-Y.values)

```

For the similarity search implementation, the following method on our code retrieves the most similar time series, along with the distance and the number of skipped original distance checking.

```

def
similaritySearch(query,query_summarized,timeseries,time
series_summarized):
    # we apply the search algorithm
    return {
'mostSimilarTimeSeriesRow' : mostSimilarTimeSeriesRow,
        'mostSimilarTimeSeries' : mostSimilarTimeSeries,
'Dclosest' : Dclosest,
'skippedQueries' : skippedQueries }

```

4. Description of pruning ratio:

Using the similarity search with summarization, some queries are skipped and the rise of the number of skipped queries means that the summarization is more efficient. To evaluate our search, we report the pruning ratio averaged on all 100 queries. Since the similarity search method returns an object that contains the number of skipped queries, all what we need to do is to calculate to call the last method for each query, and calculate the mean of skipped queries. The implementation is present in the following method:

```

def AllQueryTimeSeriesSimilaritiesPruneRatio (
all_query_timeseries,
all_query_timeseries_summarized,all_timeseries,
all_timeseries_summarized,distanceMethod ):

queriesAndSimilarity = []
queryCount = all_query_timeseries.shape[1]

for i in range(0,queryCount):

    result = similaritySearch(
all_query_timeseries(i), all_query_timeseries_summarized(i),
all_timeseries, all_timeseries_summarized, distanceMethod)
    queriesAndSimilarity.append({'query_id' : i ,

```

```

'most_similar_timeseries': result['mostSimilarTimeSeriesRow']
, 'Dclosest': result['Dclosest'] , 'skippedQueries':
result['skippedQueries'])

```

```

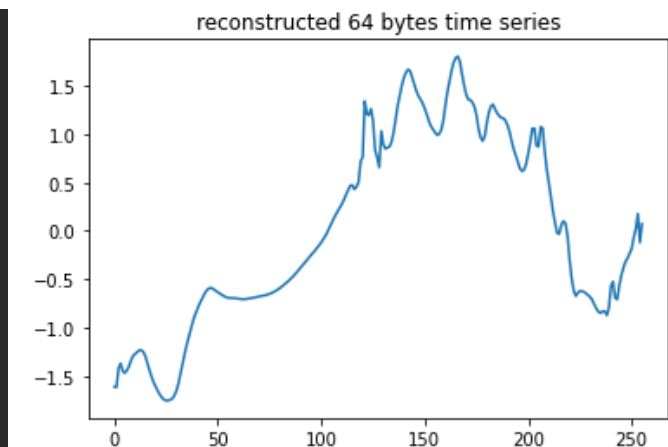
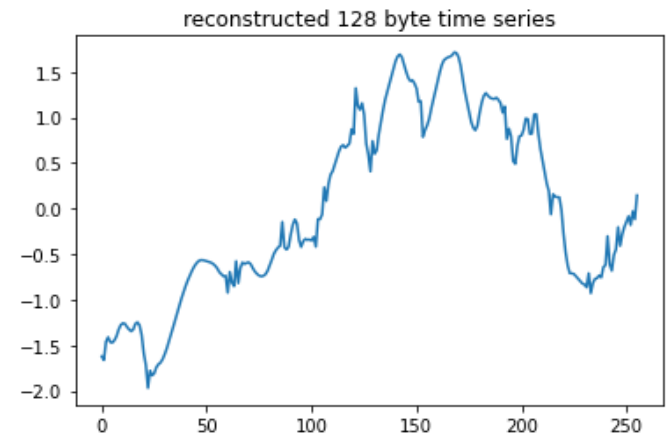
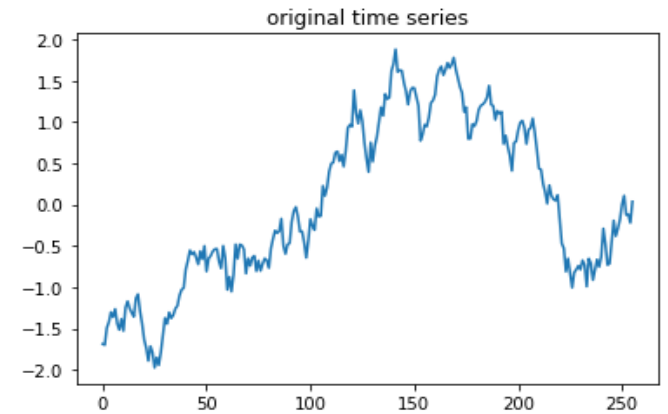
r = pd.DataFrame( queriesAndSimilarity )
return r['skippedQueries'].mean() / 50000

```

V. Result and experiment:

1. Data Reconstruction Result:

In this section we will plot some time series that have been reduced to 128, 64 and 32 bytes for both synthetic and seismic datasets. We will also give the errors for both datasets in each summarization.



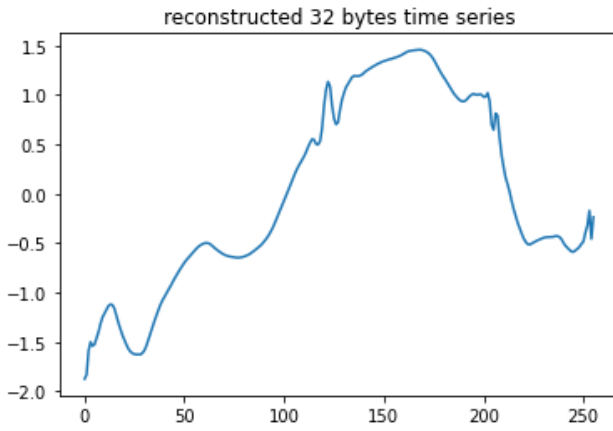


Figure 7- Synthetic Plot.

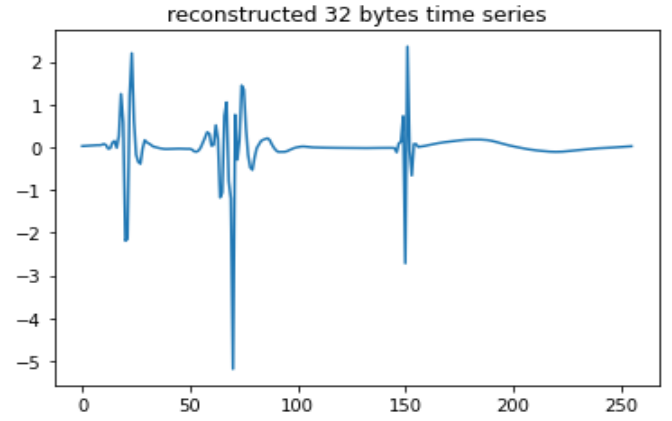


Figure 8 Seismic Plot.

From time series plot (figure 7, figure 8) we notice that the more data is reduced the more the reconstructed data is less similar to the original. We observe that the 128 bytes reconstructed time-series in both synthetic and seismic is more similar to the original.

This result is what was expected. In fact, the more we reduce our original time series the more we lose information. Meaning every time we reduce, we are producing detail coefficient that are less stored.

In the case of 128 bytes summarization, the plot is the most similar to the original one, since more data and more details coefficients are stored which is the key for a good reconstruction. Effectively, only 128 bytes of details coefficients are saved from all the 896 bytes that are generated from all the DWT level of reduction.

Meanwhile in the 64 bytes we only get to store 64 bytes from the 896 bytes of detail coefficients. Finally, only 32 bytes will be stored in the 32 bytes summarization which is the explanation for the less similarity of the reconstructed and original time series.

2. Error Result:

After calculating the error over both datasets, synthetic and seismic we obtained this result:

- Synthetic:

Mean error for 128 bytes	Mean error for 64 bytes	Mean error for 32 bytes
1.8096	2.8252	4.9204

- Seismic:

Mean error for 128 bytes	Mean error for 64 bytes	Mean error for 32 bytes
11.2092	13.9134	14.7974

The result given by the mean square evaluation, support what we explained from the description of our plot. And we can clearly observe that the 128 bytes give less error than the other summarization.

3. Similarity Search Result:

Finally, after executing the similarity method, calculating the pruning ratio on both synthetic and seismic dataset we obtain the following results.

The pruning ratio results are the following:

- Synthetic queries:

Pruning ratio using 128 bytes summarization	Pruning ratio using 64 bytes summarization	Pruning ratio using 32 bytes summarization
0.7489688	0.7489669	0.7489628

- Seismic queries:

Pruning ratio using 128 bytes summarization	Pruning ratio using 64 bytes summarization	Pruning ratio using 32 bytes summarization
0.7489688	0.7489669	0.7489628

The summarization that gives higher pruning ratio is:

- 64 bytes summarization for synthetic data.
- 128 bytes summarization for seismic data.

As shown mentioned before, the best synthetic queries pruning ratio is 0.037, meaning that each query skips on average 4 time series checking for each 100 time-series. which means that the synthetic queries are very similar when summarized. As for the seismic queries, the best pruning ratio is very high, which means that the implemented summarization is efficient as for each 100 time-series, 74 have been skipped without further checking.

VI. Conclusion:

In this project, we used DWT as summarization technique for our time series data. We discussed the use of this technique for the reduction of our original data into 3 case (128, 64 and 32 bytes), then we compared the result of reconstruction the summarized data. The result of this technique was used for similarity search.

In conclusion of this summarization we deduced that:

- The 128 bytes summarization gives the best result for reconstruction.
- Reducing using multiple DWT levels, gives us more and more detail coefficients, to ensure having a good reconstruction with less errors, we have to store more detail coefficients.

Our technique can be optimized, to give more relevant result for reconstruction by:

- Studying the type of the time series and classifying them to their proper class, this will be productive if using DWT since different type of wavelet exists.
- Developing a more efficient algorithm for storing the detail coefficients would be the key for a good reconstruction.

As for the similarity search, we conclude that:

- DWT summarization is efficient when implemented in similarity search since using it gives the same results as the original similarity search.
- For the synthetic queries, the best summarization design for a higher pruning ratio is the 64 bytes design.
- For the seismic queries, the best summarization design for a higher pruning ratio is the 128 bytes design.

VII. Bibliography:

- [1] (PDF) A Review on Time Series Dimensionality Reduction. Available from:
https://www.researchgate.net/publication/329424523_A_Review_on_Time_Series_Dimensionality_Reduction
[accessed Apr 09 2020].
- [2] <http://ataspinar.com/2018/12/21/a-guide-for-using-the-wavelet-transform-in-machine-learning/>
- [3] A. G. Ramakrishnan and S. Saha, "ECG coding by wavelet-based linear prediction," *IEEE Trans. Biomed. Eng.*, Vol. 44, No. 12, pp. 1253-1261, 1977.
- [4] <http://gwyddion.net/documentation/user-guide-en/wavelet-transform.html>
- [5] <https://www.hindawi.com/journals/aaa/2013/603629/>
- [6] Knowledge Discovery and Data Mining. Current Issues and New Applications: Current Issues and New Applications: 4th Pacific-Asia Conference, PAKDD 2000 Kyoto, Japan, April 18-20, 2000 Proceedings
- [7] https://en.wikipedia.org/wiki/Wavelet_transform#cite_note-3
- [8] <https://www.researchoptimus.com/article/what-is-time-series-analysis.php>
- [9] <https://towardsdatascience.com/an-end-to-end-project-on-time-series-analysis-and-forecasting-with-python-4835e6bf050b>