Speeding-up the Similarity Search in Time Series Databases by Coupling Dimensionality Reduction Techniques with a Fast-and-dirty Filter

Muhammad Marwan Muhammad Fuad, Pierre-François Marteau

Time Series Retrieval Using Multiple Reduced Spaces

Muhammad Marwan Muhammad Fuad
Université de Bretagne Sud
BP. 573, 56017 Vannes, France
marwan.fuad@univ-ubs.fr

***Abstract*—the similarity search problem is one of the main problems in time series data mining. Traditionally, this problem has been tackled by sequentially comparing the given query against all the time series in the database and returning the time series that are within a predetermined threshold of that query. But the large size and the high dimensionality of time series databases that are in use nowadays make that scenario inefficient. There are many representation techniques that aim at reducing the dimensionality of time series so that the search can be handled faster at a lower-dimensional space level. In this paper we propose a new method which is based on using multiple reduced spaces that correspond to certain segments' lengths. The new method also uses fast-and-dirty filters which speed up the similarity search process. For each space an approximating function is used to represent the time series at that space. The distances between the time series and those approximating functions are calculated and stored at indexing time. At query time, the designed filters use these pre-computed distances to exclude the time series which are not answers to the query, making the least number of distance computations and return a candidate response set. The representation level is increased gradually to higher levels with stronger exclusion power. The cardinality of the candidate response set gets smaller as we move from one level to another. A post-processing scanning on this response set is performed to filter out any false alarms and return the final response set. We present experimentations that compare our method with sequential scanning on different datasets, using different threshold values and different approximating functions. The experiments show that our new method is faster than sequential scanning by an order of magnitude.**

***Keywords*- Time Series Information Retrieval, Time Series Data Mining, Similarity Search, Multiple Reduced Spaces**

I. INTRODUCTION

Time series is a collection of observations at intervals of time points. These data appear in a broad variety of financial, medical, and scientific applications. Indexing, searching and retrieving of time series is one of the main tasks in time series data mining. Due to the numerous applications in which time series are involved, and the large size of time series databases, speed has always been the principal focus of all the methods and algorithms that handle this type of data.

Time series data mining deals with several tasks such as classification, clustering, similarity search, motif discovery, anomaly detection, and others. One key to performing these tasks efficiently and effectively is to use suitable indexing structures that direct the query processing towards regions in the search space, where similar time series to the query are likely to be found. This makes the retrieving process faster.

However, the high dimensionality of time series can make these indexing structures fail to handle these data. One of the best solutions to deal with the high dimensionality of time series is to utilize a dimensionality reduction technique that helps represent the time series at lower dimensional spaces, and then use an indexing structure on the reduced spaces.

Similarity between two time series can be depicted using a similarity measure. Similarity search can be viewed as retrieving all the data objects, in the repository, that are "near" a given query. This nearness can be modeled using a powerful mathematical concept; *the metric distance*, which is related to another mathematical concept; *the metric space*.

In time series data mining the most widely used distance is the Euclidean distance [12, 19], which is a metric distance. This distance is easy to compute, but it is has a few inconveniences; it is sensitive to noise and to shifts on the time axis. It is also applied to series of identical lengths only [15]

Time series representation techniques use predefined schemes. This means that the parameters which control the performance of the search algorithm have been decided at indexing time, and the performance of the search process depends highly on these parameters, which may prove to be inappropriate at query-time.

In this paper we present a new method that offers more control on these parameters. This control enables the search algorithm to use the necessary computations only, starting with the less costly computations, whose excluding power is lower, and moving to more expensive computations, with more exclusion power, only when less costly computations fail to exclude the time series

The rest of the paper is organized as follows: in section 2 we present the necessary background, and main concepts of similarity search, in general, and search in time series databases, in particular. Our method is explained in section 3. The experimental part is presented in section 4. Section 5 discusses the results of our experiments, and finally in section 6 we present a conclusion and the different directions of future work..

II. BACKGROUND

A. *Materic Spaces*

Let D be a set of objects. A function

$d: D \times D \rightarrow \Re^{+} \bigcup \{0\}$, is called a distance metric if the following holds;

i- $d(x, y) \geq 0$ (non-negativity)

ii- $d(x, y) = d(y, x)$ (symmetry)

iii- $x = y \Leftrightarrow d(x, y) = 0$ (identity)

iv- $d(x, z) \leq d(x, y) + d(y, z)$ (triangular inequality)

$\forall x, y, z \in D$. We call (D, d) a metric space

Search in metric spaces has many advantages, the most common of which is that a single indexing structure can be applied to several kinds of queries and data types, which can be very different in nature. This is mainly important in establishing unified models for the search problem that are independent of the data type. This makes metric spaces a solid structure that is able to handle several data types.

In practice, distance metrics are not always easy to find or to apply. Many similarity search paradigms relax some of the above conditions and use similarity measures instead. Similarity measures can be viewed as a weak form of the distance metric. The most important condition of the four above ones is the triangular inequality, because by using this inequality the search algorithm can exclude data objects, which are not answers to the query, without making unnecessary distance computations. This speeds up the search process, because distance computation is usually the most costly operation.

There are so many distance functions that are known in the multimedia community, some of them are general, while others are used with certain data types only. The most common distance is the *Minkowski distance*: This is actually a whole family of distances, designated by L_P .This distance is defined in a n-dimensional space as:

$$L_P[(x_1, x_2, ..., x_n), (y_1, y_2, ..., y_n)] = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \qquad (1)$$

If $p = 1$, the distance is called the *Manhattan distance* or the *city block distance*. If $p = \infty$, it is called the *infinity distance* or the *chessboard distance*. And if $p = 2$,we get the well-known *Euclidean distance*.

B. Range Queries

Given a query q and a radius r , which represents a *threshold*, a *tolerance*, or a *selectivity*. The range query problem can be specified as retrieving all the data objects that are within a distance r of that query. This can be represented as:

$$R(q, r) = \{u \in U; d(q, u) \leq r\} \qquad (2)$$

Where U is the set of objects in the database.

Range queries have a main drawback: in some cases we do not have any prior knowledge about the database in question. So assigning an inappropriate value to r may sometimes result in two undesirable situations; returning a response set that is too large, or returning an empty response set. What happens in these cases is that the user restarts the query using a different value of r , and this may happen several times before getting a satisfying size of the response set. In very large databases, with a computationally expensive distance function, this can be laborious.

C. Sequential Scanning

In many applications distance computations can be a time consuming task that other tasks such as CPU time or even I/O time can be neglected. For this reason search algorithms try to avoid distance computations as much as possible. In fact, in many cases the performance of an algorithm is measured by the number of distance computations it requires. Different distances also take different computing times. For instance, the *dynamic time warping* (DTW), which is a similarity measure that is widely used in time series databases, gives better results in time series data mining tasks than the Euclidean distance, but it is more costly to compute.

A trivial solution to the similarity search problem is sequential scanning, also known as *linear scanning*, where the query is compared against all the data objects in the data base. So in order to perform a range query $R(q, r)$, the distance between the query q and all the data objects in the data base is computed, and all the data objects that satisfy $d(q, u) \leq r$ constitute the response set.

It is easy to notice that in the cases where the size of the data base is very large, which is the case with most databases in use today, sequential scanning is not the best scenario to answer similarity queries because it requires too many distance evaluations.

Different techniques can be used to avoid the high cost of sequential scanning. One of them is using indexing structures, which are offline procedures based on storing some distance calculations between all the data points and other points, usually called pivots, in different data structures (trees, partitions, etc) . Later, and at query time, these calculations can be used to exclude some data objects, which, according to these pre-computed distances, can not be answers to the query. This is what we call a *fast-and-dirt filtering* of data. What remains of the data objects is scanned sequentially against the query to get the true answers to the query.

Nevertheless, even these structures can fail in handling high-dimensional databases that their performance can deteriorate to become similar to that of sequential scanning, or even worse. This is what we call the *dimensionality curse*.

D. The Dimensionality Curse

The term “dimensionality curse”, which is also known by “Hughes effect”, was first introduced in [3]. It refers to the fact that in order to estimate a function of several variables to a given accuracy, the number of data samples required grows exponentially as the number of dimensions grows linearly [14]. High dimensional data spaces are inherently sparse, because available data are usually limited. This results in what it known as the *empty space phenomenon* [21]

What makes high dimensional spaces difficult to handle is that most of the effects resulting from the increase of

dimensionality are unintuitive. While our perception of high dimensional spaces is based on extending our image of low dimensional spaces, this technique becomes deceptive when the dimension of the space becomes high, which is a part of the curse of dimensionality effect.

Another result of the dimensionality curse is that by increasing space dimensionality more space is required to store a single data object. As a consequence, the index fanout (the number of children per node) is reduced considerably, resulting in an increase in disk accesses. In addition, the good properties of index structures no longer hold because of the excessive overlap, hence the discrimination power of the structure decreases [18]

E. Representaion Techniques

Time series are highly correlated data, so one of the widely-known schemes to handle these data is projecting the original time series onto lower dimensional spaces and processing the query in those reduced spaces.

When embedding the original space into a lower dimensional space and performing the similarity query in the transformed space, two main side-effects may be encountered; *false alarms* and *false dismissals*. False alarms are data objects that belong to the response set in the transformed space, but do not belong to the response set in the original space. False dismissals are data objects that the search algorithm excluded in the transformed space, although they are answers to the query in the original space. Generally, false alarms are more tolerated than false dismissals, because a post-processing scan is usually performed on the results of the query in the transformed space to filter out these data objects that are not valid answers to the query in the original space. However, false alarms can slow down the search time if they are too many. False dismissals are a more serious problem and they require more sophisticated procedures to be avoided.

False alarms and false dismissals are dependent on the transformation used in the embedding. If f is a transformation from the original space (S_{orig}, d_{orig}) into another space (S_{trans}, d_{trans}) then in order to guarantee no false dismissals this transformation should satisfy:

$$d_{trans}(f(u_1), f(u_2)) \le d_{orig}(u_1, u_2), \quad \forall u_1, u_2 \in S_{orig} \quad (3)$$

The above condition is known as the *lower-bounding lemma*. [22]

The GEMINI Framework: In [8] the authors presented a generic approach for indexing time series. Later GIMINI was extended to other data types. GEMINI reduces the dimensionality of the time series by converting them from a point in a *n*-dimensional space into a point in a *m*-dimensional space, where $m<<n$. A similarity distance is defined on the reduced space, which is lower bounding to the original similarity distance, thus the similarity search returns no false dismissals. A post-processing scan on the candidate response set is performed to filter out all false alarms and return the final response set. Table 1 illustrates the GEMINI algorithm.

TABLE I. THE GEMINI ALGORITHM FOR TIME SERIES RANGE QUERIES

Algorithm: range_query(Q,r)

1. Transform the time series in the database DB from the original n-dimensional space into a lower dimensional space of m dimensions
2. Define a lower bounding distance on the reduced space: $d^m(S_i, S_j) \le d^n(S_i, S_j) \ \ \forall S_i, S_j \in DB$
3. Eliminate all the time series for which we have $d^m(Q,S) > r$ → obtain a candidate response set
4. Apply d^n to the candidate response set and eliminate all the time series that are farther than r from Q to get the true response set.

Research in time series data mining has focussed on these two aspects: dimensionality reduction methods, and similarity distances.

There have been different suggestions to represent time series in lower dimensional spaces. To mention a few: *Discrete Fourier Transform* (DFT) [1, 2], *Discrete Wavelet Transform* (DWT) [5], *Singular Value Decomposition* (SVD)[13], *Adaptive Piecewise Constant Approximation* (APCA) [11], *Piecewise Aggregate Approximation* (PAA) [10,22] , *Piecewise Linear Approximation* (PLA) [16], and *Chebyshev Polynomials* (CP) [4].

Different similarity distances have also been used. While the Euclidian distance is the most widely-known distance in the literature, other distances have also been used, one of which is the DTW we mentioned in section 2.3. Other distances are *Edit distance with Real Penalty* (ERP) [6], *Edit distance with Real Sequence* (EDR) [7], and the *Extended Edit Distance* (EED) [17].

III. THE PROPOSED METHOD

A. Motivation

Time series representation methods have the following scheme, choose a lower-dimensional space, project the data objects on that space, define a lower bounding distance function or similarity measure on the reduced space , process the similarity search in the reduced (transformed) space , exclude the data objects that are farther than r from the query. At the end, we get a candidate response set, because, as we mentioned above, this scheme may produce some (few or many, depending on the method) false alarms, so a post-processing scan on this candidate answer set, using the original data objects and the original distance function or similarity

measure, is performed to get the final answer set. Pre-computed methods use a similar scheme.

The problem with all these methods is that they are a one-step method. They decide at indexing-time, the dimension of the transformed space. And the performance at query-time depends completely on the choice of dimension of the transformed space. But in practice, we do not know a priori the optimal dimension of the reduced space or the optimal number of pivots.

In this paper we try to address this problem in a different way that involves a multi-resolution representation of time series; we use *multiple reduced spaces*, or as we call them *resolution levels* and they store different numbers of pre-computed distances. Lower resolutions have lower dimensions, so distance calculations are less costly than higher resolutions, where dimensions are higher, so distance calculations are more expensive. But the distances that we compute at any level are always less expensive than the distances used in sequential scanning, because even at the highest level, the dimension is still lower than that of the original space, which is used in sequential scanning. In our method, the search algorithm starts with the lowest level, and tries to exclude the data objects, which are not answers to the query, at that level, where the distances are not costly to compute, and it does not access a higher level until all the pre-computed distances of the lower level have been exploited. Our method uses a strategy that economizes distance computations to the lowest degree possible.

B. *The poposed Method*

Let O be the original, n-dimensional space where the time series are embedded, R is another space, whose dimension is $2m$, where $2m \le n$. Each time series $S \in O$ is segmented into m segments. Each segment $[t_i,t_j]$ of this time series is approximated by a function of low dimension: a polynomial of degree (1:5), for instance, where the degree of this approximating function is lower than the length (the number of points) of that segment, and where the approximation error, according to a given distance, between this segment and the approximating function is minimal, so this function is the best approximation of that segment. A polynomial of the same degree is used to approximate all the segments of all the time series of the database.

We associate every segment with two related concepts; the first is the image of all the points of that segment on the approximating function. The *image vector* $\tilde{S}$ is, by definition, a n-dimensional vector whose components are the images of all the points of all the segments of that time series. The second concept is the images of the two end points of each segment $[t_i, t_j]$ on that approximating function, which we call the *main image* of that segment. So for a time series of m segments we have $2m$ main images. Those $2m$ main images are, by definition, the *projection vector* S^R of the time series on R.

Figure 1 illustrates the different definitions we presented in this section. The segment [0:3] is approximated by a first-degree polynomial. The image of this segment is the points [*a*, *b*, *c*, *d*]. The main image of this segment is the points [*a*, *d*].

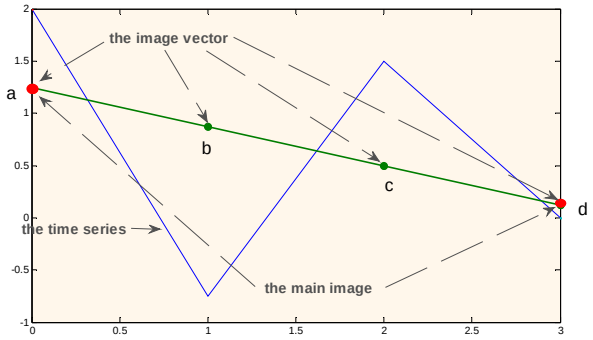

Figure 1. The different concepts of the proposed method

We define two distances on the database. The first is denoted by d, and is defined on an n-dimensional space, so it is the distance between two time series in the original space, i.e. $d(S_i,S_j)$, or the distance between the original time series and its image vector, i.e. $d(S_i,\tilde{S}_i)$. The second distance is denoted by d^R, and is defined on a $2m$-dimensional space, so it is the distance between two projection vectors, i.e. $d^R(S_i^R,S_j^R)$.

Notice that since the main image of each segment is a partial set of the image of that segment, this implies that d^R is a partial distance of d. The direct result of this is that if we use the Euclidean distance (or any Minkowski distance), for both d and d^R we get:

$$d^R(S_1^R,S_2^R) \le d(\tilde{S}_1,\tilde{S}_2) \qquad (4)$$

Relation (4) means that d^R is a lower bounding of d.

The *resolution level* $k \in \aleph$ is a number related to the dimensionality of the reduced space R. So the above definitions of the projection vector and the image vector can be extended to further segmentation of the time series, with different values $m \le m_k$, The image vector and the projection vector at level k are denoted by $\tilde{S}^{(k)}$ and $S^{R(k)}$, respectively. Figure 2 shows an example of the relationships between the previous concepts.

C. *The Double Filtering Inequalities*

Given a range query Q and a radius r.By applying the triangular inequality we get:

$$d(\tilde{Q}^{(k)},S) \le d(Q,S) + d(Q,\tilde{Q}^{(k)}) \quad \forall S \in O \qquad (5)$$

So now the range query can be expressed as :

$$d(\tilde{Q}^{(k)},S) \le r + d(Q,\tilde{Q}^{(k)}) \qquad (6)$$

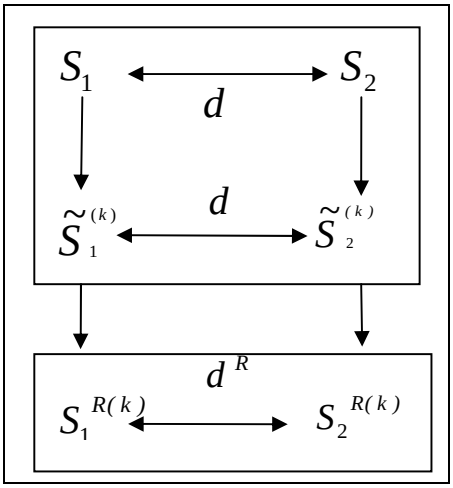

Figure 2. The original space *(O,d)* embeds the original time series S_1, S_2 (top), the images of the approximated time series $\tilde{S}_1$, $\tilde{S}_2$ (middle). The reduced space R embeds the main images of the approximated time series S_1^R, S_2^R (bottom)

Since $\tilde{S}^{(k)}$ is the best approximation of S at level k , then for any $S \in O$ we have:

$$d(\tilde{Q}^{(k)}, S) \geq d(S, \tilde{S}^{(k)}) \tag{7}$$

So (6) can be expressed as:

$$d(S, \tilde{S}^{(k)}) \leq r + d(Q, \tilde{Q}^{(k)}) \tag{8}$$

This means that all the data objects that satisfy:

$$d(S, \tilde{S}^{(k)}) > r + d(Q, \tilde{Q}^{(k)}) \tag{9}$$

Should be excluded.

In a similar way, by applying the triangular inequality, we have:

$$d(Q, \tilde{S}^{(k)}) \leq d(Q, S) + d(S, \tilde{S}^{(k)}) \tag{10}$$

And the range query can be expressed as :

$$d(Q, \tilde{S}^{(k)}) \leq r + d(S, \tilde{S}^{(k)}) \tag{11}$$

Since $\tilde{Q}^{(k)}$ is the best approximation of Q at level k , then for any $S \in O$ we have:

$$d(Q, \tilde{S}^{(k)}) \geq d(Q, \tilde{Q}^{(k)}) \tag{12}$$

So (11) can be expressed as:

$$d(Q, \tilde{Q}^{(k)}) \leq r + d(S, \tilde{S}^{(k)}) \tag{13}$$

This means that all the data objects that satisfy:

$$d(Q, \tilde{Q}^{(k)}) > r + d(S, \tilde{S}^{(k)}) \tag{14}$$

Should be excluded

From both (9) and (14), we get:

$$\left| d(Q, \tilde{Q}^{(k)}) - d(S, \tilde{S}^{(k)}) \right| > r \tag{15}$$

Inequality (15) defines the first exclusion condition, which we call *the first filter*

On the other hand, by using the triangular inequality, we have :

$$d(\tilde{S}^{(k)}, \tilde{Q}^{(k)}) \leq d(\tilde{Q}^{(k)}, S) + d(S, \tilde{S}^{(k)}) \tag{16}$$

Using the triangular inequality again, and substituting in the above relation we get:

$$d(\tilde{S}^{(k)}, \tilde{Q}^{(k)}) \leq d(Q, S) + d(Q, \tilde{Q}^{(k)}) + d(S, \tilde{S}^{(k)}) \tag{17}$$

Or:

$$d(\tilde{S}^{(k)}, \tilde{Q}^{(k)}) \leq r + d(Q, \tilde{Q}^{(k)}) + d(S, \tilde{S}^{(k)}) \tag{18}$$

If d and d^R are Euclidean, and taking (4) into account, we can write:

$$d^R(S^{R(k)}, Q^{R(k)}) \leq d(\tilde{S}^{(k)}, \tilde{Q}^{(k)}) \tag{19}$$

By substituting in (18) we get the second exclusion condition:

$$d^R(S^{R(k)}, Q^{R(k)}) > r + d(Q, \tilde{Q}^{(k)}) + d(S, \tilde{S}^{(k)}) \tag{20}$$

We call the above exclusion condition *the second filter*

D. The Algorithm

At indexing time: We start by choosing the length of segments at each resolution level. The segments at the lowest resolution level are the longest. The length of segments gets shorter as the resolution level gets higher. Then we choose the approximating function to be used with all the time series and for all resolution levels. This means that if we choose to approximate the time series by a first-degree polynomial, all the time series in the database should be approximated by a first-degree polynomial.

We compute and store all the values $d(S, \tilde{S}^{(k)}) \quad \forall S \in O$

At query time: The query is segmented at each resolution level using the same length of segments that was used to segment the time series. Then these segments are approximated using the same approximating function that was used to approximate the time series. The distances $d(Q,\tilde{Q}^{(k)})$ are computed. Notice that $d(Q,\tilde{Q}^{(k)})$ is computed only once for all the time series in the database.

At each resolution level, filter one is much less costly to apply than filter two, because it does not include any distance computations, since the two distances it uses have already been computed at indexing time. Filter two contains two distances that have been computed at indexing time: $(d(Q,\tilde{Q}^{(k)}),d(S,\tilde{S}^{(k)}))$, so the only distance that is to be computed at query time is $d^R(S^{R(k)},Q^{R(k)})$. Since lower resolution levels have lower dimensions, filter two is less costly to compute at those levels than at higher levels, where the dimensionality increases. But at any level, the cost of computing filter two is never as costly as distance calculations at the original space, because we assumed that $2m \le n$.

The algorithm starts at the lowest resolution level by applying filter one to the first time series. If this time series is excluded, we move to the next time series and apply filter one to this time series, if not, the algorithm applies filter two to the first time series, then it moves to the next time series. At any stage, if all the time series have been excluded the algorithm terminates immediately. Only when all the time series have been examined at a certain resolution level does the algorithm move to a higher level. In this higher level filter two is more costly than it was at the lower level, so our algorithm does not compute a more expensive distance calculation unless it has tried to exclude the time series using a less expensive distance at a lower resolution level.

Applying the two filters , and moving from one resolution level to a higher one produces a response set, whose cardinality becomes smaller and smaller.

At the end, and after all resolution levels have been used , we get a candidate response set, which contains all the true answers to the query, since our method uses a lower bounding distance, but it could also contain some false alarms. This answer set is post-processed to get the final answer set.

IV. EXPERIMENTS

We tested our new method using datasets available at UCR [23]. To see how our method scales, we chose the datasets with the largest sizes. These datasets are: (50words), (CBF), (FaceAll), (Two_Patterns), (wafer), and (yoga). The size of these datasets varies between 900 (CBF), and 6164 (wafer). We used segment lengths that are of the power of 2. For example, when the length of the time series is 128 (CBF, Two_Patterns), the length of each segment at the first resolution level is 64, and at the second it is 32, and so on. So we obtained 7 resolution levels with (50words) and (yoga) and 6 resolution levels with the others. The approximating function we used was of the first degree in the first experiment (linear fitting), of the third degree in the second experiment and of the fifth degree in the third experiment. The distance we used for both d and d^R was the Euclidian distance. We compared our method with sequential scanning. For all the datasets, the values of r varied between r that returns 1% of the time series of that dataset (in sequential scanning) and r that returns 10% of the time series

For each data set and for each value of r we launched the query 100 times and took the average of these 100 runs. The queries in all cases were time series in that dataset chosen at random, and then noise was added to them.

We decided to a use a platform-independent approach to test our method using *latency time*, so we added a counter to compute the number of different operations (>,+ - ,*,abs, sqrt) that both sequential scanning and our method used in the search

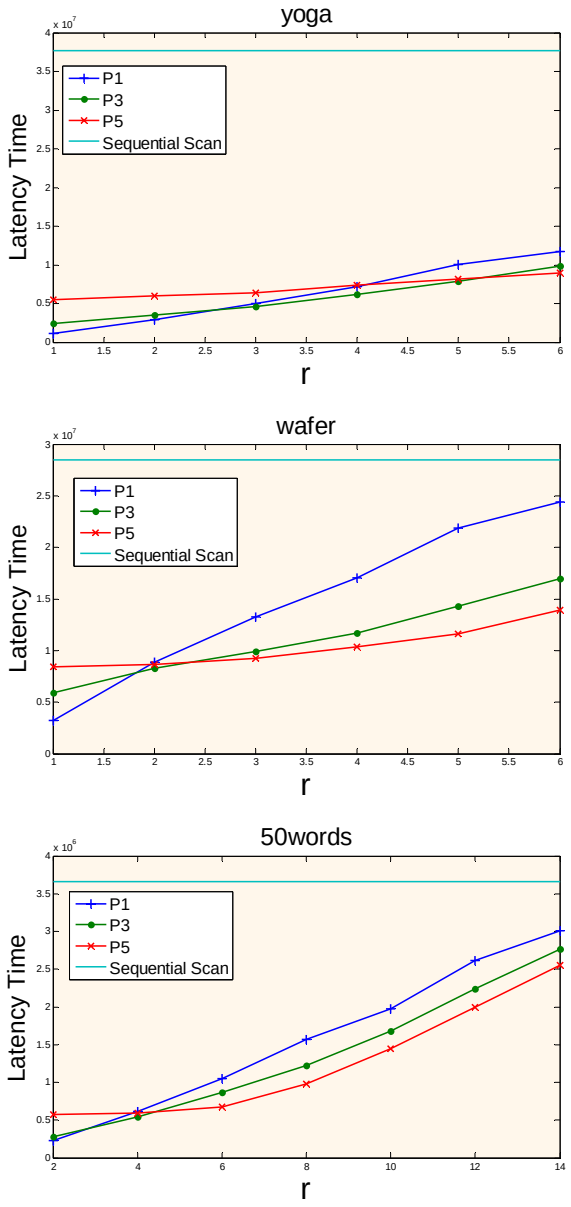

Figure 3. Comparison of the latency times of sequential scanning (dashed line) and the latency time of our method on datasets (yoga) (above), (wafer) (middle) and (50words) (below). The figure shows the latency time using as an approximating function a first degree polynomial (P1), a third degree polynomial (P3), and a fifth degree polynomial (P5)

process. Then the number of each operation was multiplied by the latency time of that operation to get the total latency time for sequential scanning and for our method. The latency time of the different operations was obtained from a performance study of floating point operations [20]. According to this study, the latency time (given in processor cycles) is 5 cycles for (>, + -), 1 cycle for (abs), 24 cycles for (*), and 209 cycles for (sqrt). This approach actually puts our method at a disadvantage, because our method uses the square root operation, which is an expensive operation, more often. So the results of our method should be viewed as a worst case performance of our method. Figure 3, shows the results we obtained using as an approximation function a first, third, and fifth degree polynomials. Due to space limitation, we present the results of three datasets only: (yoga), (50words), and (wafer). The results show that our method outperforms sequential scanning by an order of magnitude on average.

V. Discussion

The experiments show that the general performance of our algorithm outperforms that of sequential scan. The performance of the two filters seems to be complementary, since filter one filters out more time series at lower resolution levels, while filter two filters out more time series at higher levels. This phenomenon can be explained by examining the relations (15) and (20): at lower resolution levels segments are longer, so the approximation error is higher. As a consequence, the absolute difference in filter one is a difference between relatively large numbers: $d(Q,\tilde{Q}^{(k)})$, $d(S,\tilde{S}^{(k)})$,so this difference has a better chance of exceeding the value of r and excluding the time series than at higher levels, where the approximation is better, so these numbers become smaller and their difference has less chance of exceeding r

The performance of filter two is different; at lower levels $d^R(S^{R(k)},Q^{R(k)})$ is small while $d(Q,\tilde{Q}^{(k)})+d(S,\tilde{S}^{(k)})$ is relatively large (see the beginning of this section) , so the chance for this filter to exclude time series is low. As the resolution level gets higher $d^R(S^{R(k)},Q^{R(k)})$ gets bigger, while $d(Q,\tilde{Q}^{(k)})+d(S,\tilde{S}^{(k)})$ gets smaller, so this filter has a better chance of excluding time series at higher levels.

Another thing we can notice is that the performance of the algorithm gets better as the degree of the approximating function gets higher, which is easy to explain, since the approximation error gets smaller. An interesting phenomenon is that for very small r, the performance drops as we use a higher degree approximating function. We think the reason for this is that the algorithm pays an overhead cost when using a higher degree approximation.

We also notice that as the degree of the approximating function gets higher, the performance of the first filter deteriorates as shown in Table 2 and Figure 4.

VI. Conclusion and Future Work

In this paper we presented a new method that uses a new paradigm to tackle the similarity search problem. The conducted experiments give promising results compared with those obtained by sequential scanning. In all experiments, the performance was much better than that of sequential scanning for small values of r , and was still better than sequential scanning, even for large values of $r$levels. This phenomenon can be explained by examining the relations (15) and (20): at lower resolution levels segments are longer, so the approximation error is higher. As a consequence, the absolute difference in filter one is a difference between relatively large numbers: $d(Q,\tilde{Q}^{(k)})$, $d(S,\tilde{S}^{(k)})$,so this difference has a better chance of exceeding the value of r and excluding the time series than at higher levels, where the approximation is better, so these numbers become smaller and their difference has less chance of exceeding r

In this paper we presented the results obtained by using the Euclidean distance, but our method can support a variety of distances. We conducted several preliminary experiments using other distances and they gave similar results.

TABLE II. Filtering efficiencies (in percent) as a function of the fitting polynomial and the resolution level for the 50words dataset, for range queries that retrieve 1% (top) and 10% (bottom) of this dataset. The numbers are rounded-up.

		N/128	N/64	N/32	N/16	N/8	N/4	N/2	AVR
P=1	Filter 1	40	34	13	1	0	0	0	89
	Filter 2	0	0	0	2	9	1	0	11
P=3	Filter 1	64	15	2	0	0	0	0	81
	Filter 2	0	0	4	14	1	0	0	19
P=5	Filter 1	74	6	0	0	0	0	0	80
	Filter 2	0	2	12	6	0	0	0	20

		N/128	N/64	N/32	N/16	N/8	N/4	N/2	AVR
P=1	Filter 1	0	2	1	0	0	0	0	3
	Filter 2	0	0	0	0	1	74	20	94
P=3	Filter 1	5	2	0	0	0	0	0	7
	Filter 2	0	0	0	0	9	71	11	90
P=5	Filter 1	6	0	0	0	0	0	0	6
	Filter 2	0	0	0	0	13	67	11	91

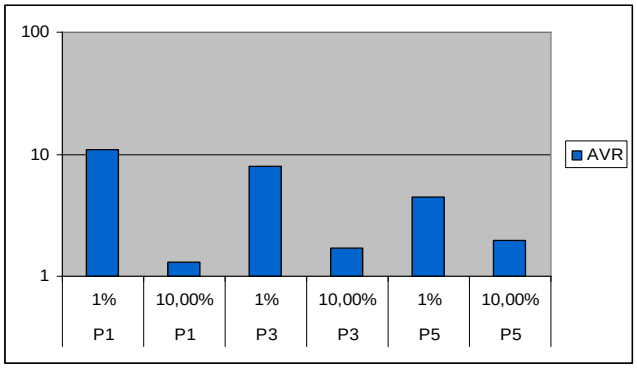

Figure 4. Speedups for queries that retrieve 1% and 10% of the datasets averaged on the 6 experimented datasets (50words, CBF, FaceAll, wafer, Two_patterns and yoga).

While working on this method and conducting the experiments we realized that there are many heuristics that can be used to improve this method. The first direction of improvement is optimizing the filtering process by sorting the distances before filtering. The preliminary experiments that we conducted using some optimising heuristics, which we did not report in this paper because they are not complete yet, showed that sorting improved the performance of our method. Other directions of optimizing, like recycling some calculations from lower levels, gave good results too, yet we need to find the best recycling scheme. The approximating functions we used in this paper were polynomials, but we think that other types of functions, which are particularly designed to approximate time series, can even give better results. We notice that the performance of the two filters depends highly on the resolution level. This idea can be exploited by using different segmenting schemes, mainly schemes that use finer segmenting in the areas where the approximation error is large, and a coarser segmenting in the areas where the approximating error is small.

The fact that filter one stops filtering out time series at a certain level and that filter two starts filtering out at a certain level can suggest an "economised" scheme for applying the filters.

Another, and more important, direction of future work is to use one of the dimensionality reduction methods, which are well-known in the literature, as an approximating function. The principle here is that dimensionality reduction methods aim at finding the best representation of the original space at a lower space, so this can be viewed as an approximation of the time series.

When we started developing our method, our aim was also to use it on multidimensional time series, so this is still a main direction of future work.

The final direction of future work is to apply our method to other data types, where the idea of resolution level is pertinent.

REFERENCES

[1] Agrawal, R., Faloutsos, C., & Swami, A.: Efficient Similarity Search in Sequence Databases". Proceedings of the 4th Conf. on Foundations of Data Organization and Algorithms (1993)

[2] Agrawal, R., Lin, K. I., Sawhney, H. S. and Shim, K.: Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-series Databases. In Proceedings of the 21st Int'l Conference on Very Large Databases. Zurich, Switzerland, pp. 490-501(1995).

[3] Bellman, R. : Adaptative Control Processes: A Guided Tour. Princeton University Press, Princeton, NJ (1961).

[4] Cai, Y. and Ng, R. : Indexing Spatio-temporal Trajectories with Chebyshev Polynomials. In SIGMOD (2004).

[5] Chan, K. & Fu, A. W.: Efficient Time Series Matching by Wavelets. In proc. of the 15th IEEE Int'l Conf. on Data Engineering. Sydney, Australia, Mar 23-26. pp 126-133 (1999).

[6] Chen, L., and Ng, R..: On the Marriage of Edit Distance and Lp norms. In VLDB, (2004).

[7] Chen, L., Ozsu, M. T. and V. Oria. V.: Robust and Fast Similarity Search for Moving Object Trajectories. In SIGMOD, (2005).

[8] Faloutsos, C., Ranganathan, M., & Manolopoulos, Y. : Fast Subsequence Matching in Time-series Databases. In Proc. ACM SIGMOD Conf., Minneapolis (1994)

[9] Jessica Lin, Eamonn J. Keogh, Stefano Lonardi, Bill Yuan-chi Chiu: A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. DMKD (2003).

[10] Keogh, E,. Chakrabarti, K,. Pazzani, M. & Mehrotra: Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. J. of Know. and Inform. Sys. (2000).

[11] Keogh, E,. Chakrabarti, K,. Pazzani, M. & Mehrotra: Locally Adaptive Dimensionality Reduction for Similarity Search in Large Time Series Databases. SIGMOD (2001).

[12] Keogh, E. & Kasetty, S.. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. July 23 - 26, 2002. Edmonton, Alberta, Canada. pp 102-111(2002)

[13] Korn, F., Jagadish, H & Faloutsos. C.: Efficiently Supporting ad hoc Queries in Large Datasets of Time Sequences. Proceedings of SIGMOD '97, Tucson, AZ,pp 289-300 (1997).

[14] Lee, J.A. and M. Verleysen, M. : Nonlinear Dimensionality Reduction. Springer (2007).

[15] Megalooikonomou, V., Wang, Q., Li, G. & Faloutsos, C. Multiresolution Symbolic Representation of Time Series. In proceedings of the 21st IEEE International Conference on Data Engineering (ICDE). Tokyo, Japan. Apr 5-9. (2005).

[16] Morinaka, Y., Yoshikawa, M. , Amagasa, T., and Uemura, S.: The L-index: An indexing Structure for Efficient Subsequence Matching in Time Sequence Databases. In Proc. 5th PacificAisa Conf. on Knowledge Discovery and Data Mining, pages 51-60 (2001).

[17] Muhammad Fuad, M.M. and Marteau, P.F.: Extending the Edit Distance Using Frequencies of Common Characters, 19th International Conference on Database and Expert Systems Applications (DEXA'08), Turin, Italy,2008 . Lecture Notes in computer Science (LNCS). Springer Berlin / Heidelberg, Vol. 5181. pp 150-157 (2008)

[18] Papadopoulos,A.N. and Y. Manolopoulos, Y.: Nearest Neighbor Search: A Database Perspective. Springer-Verlag, New York, (2005).

[19] Reinert, G., Schbath, S. & Waterman, M. S. Probabilistic and Statistical Properties of Words: An Overview. Journal of Computational. Biology. Vol. 7, pp 1-46. (2000)

[20] Schulte,M.J. ,Lindberg, M. and Laxminarain, A. :Performance Evaluation of Decimal Floating-point Arithmetic in IBM Austin Center for Advanced Studies Conference, February (2005).

[21] Scott, D.W and J.R. Thompson, J.R : Probability Density Estimation in Higher Dimensions. In J.R. Gentle, editor, Proceedings of the Fifteenth Symposium on the Interface, pages 173–179. Elsevier Science Publishers, B.V., North-Holland, (1983).

[22] Yi, B,K., & Faloutsos, C.: Fast Time Sequence Indexing for Arbitrary Lp norms. Proceedings of the 26st International Conference on Very Large Databases, Cairo, Egypt (2000).

[23] UCR Time Series datasets
http://www.cs.ucr.edu/~eamonn/time_series_data/