A Novel Similarity Search Approach for Streaming Time Series

Yiming Ding¹, Wei Luo¹, Yufei Zhao², Zhen Li¹, Peng Zhan¹,², and Xueqing Li¹,²,³

¹School of Software, Shandong University, Jinan, China
²School of Aerospace Engineering, Tsinghua University, Beijing, China
³Email: zhanpeng@sdu.edu.cn
⁴Email: xqli@sdu.edu.cn
※Corresponding Author

Abstract. Time series similarity search has been widely used in many applications, such as financial data analysis, meteorological data forecasting, and multimedia data retrieval. The original task of this research was to find time series in a database similar to the query sequence, where both the results and the query sequence are static. However, along with the arrival of Industry 4.0 era, a large number of detecting instruments in various fields have produced a large amount of time series stream data. The new challenges for Similarity-based time series retrieval can be attributed as one-pass search and real time response. In this paper, we propose a novel multi-resolution search scheme to find a set of sequences similar to the query sequence in streaming time series efficiently and exactly. The scheme performs multi-resolution filtering on the raw data, and most dissimilar sequences can be filtered out at a lower level to avoid unnecessary computation and to improve the search efficiency. Extensive experiments on different kinds of typical time series datasets have been conducted to demonstrate the superiorities of our method.

1. Introduction

Along with the arrival of Industry 4.0 era, extensive detecting instruments in various fields have produced a large amount of streaming time series [1, 2]. Compared to traditional time series, time series stream data is continuously generated which leads to updating frequently and impossible to be stored as a whole in the memory or the hard disk [3, 4]. As a result, similarity-based time series stream data retrieval has brought new challenges, that is, "One-pass scanning" processing and fast response. The previous methods applied to traditional time series may not work well in this scenario due to their limited capability on processing the stream data update [5].

There are several types of scenarios occurring about similarity search in time series stream data. These scenarios can have either streaming or static queries, and the database either is fixed or consists of data streams. In prior related work [6, 7], only the scenario with streaming queries and fixed time series database is discussed. However, in many important applications, the database itself is consists by streaming data too. For example, in stock market analysis, a database is usually formed by incrementally storing multiple data streams of stock prices [8].

Accordingly, we decide to deal with an important scenario in stream applications where incoming data is from multiple continuous stream time series. At each time stamp, a new data item is appended to the stream time series. We detect which subsequences to the current time are similar to the query one so that their distances do not exceed a user-specified threshold $\varepsilon$.

In time series streams, time series arrive at high speed, and therefore we need to process queries very fast. Index structures have been effectively employed in traditional databases to improve the
query performance, such as R-tree based index structures, but they are not suitable for indexing data streams since they are designed for the cases where the dimensionality is fixed. A multi-step filtering mechanism, named MSM for short, is used to do similarity match over time series streams in paper [9]. The mechanism can greatly prune the search space and thus offer fast response. However, if we represent the time series with MSM, we have to fix the length of patterns to integer multiple of 2.

In this paper, we propose a novel scheme based on multi-resolution filtering to perform the similarity search in streaming time series scenario efficiently and exactly, which can be named MRSS. Our scheme doesn’t fix the length of the query sequence and provides a more accurate multi-resolution filtering scheme based on the shape of the query sequence. During the multi-resolution filtering process, the distance between query sequence and stream time series is calculated incrementally, and most false candidates can be filtered out at a lower level before computing the real distance to save computation cost and offer faster response. Finally we compare our MRSS method for similarity range query on extensive real open source datasets to show the advantages of our approach.

2. Methodology

When we represent a time series approximately in multi-resolution, the multi-resolution matching process is that filter the data object at a low resolution, and if it cannot be filtered, filter it at a higher resolution until the highest resolution or a certain termination condition is reached.

2.1. Time Series Multi-Resolution Dimensionality Reduction

The initial approximation of the time series is the mean of the segments between the start and end points. After that, we obtain multi-resolution dimensionality reduction by adding the feature points to the initial segment one by one according to the weights from large to small, and then the number of segments representing the time series is increased so that the representation of the time series is more and more accurate. The total number of feature points is the total number of levels represented by multi-resolution.

Assuming that there is a time series \( Q \) with \( n \) feature points. The multi-resolution approximation of \( Q \) is expressed as: \( A(Q) = [A_1(Q), A_2(Q), ..., A_n(Q)] \), where \( A_i(Q) \) is an approximate representation of the \( i^{th} \) resolution of the time series \( Q \), \( 1 \leq i \leq n \). At the \( i^{th} \) resolution, the time series is divided into \( i+1 \) segments by the \( i \) feature points, and \( A_i(Q) \) is composed of the mean of each segment, ie \( A_i(Q) = [m_1, m_2, ..., m_{i+1}] \), \( m_i \) is the mean of the \( i^{th} \) segment. If we save \( A(Q) \) with an array, the number of the data items that we need to save is \( \sum_{i=2}^{n+1} i = n(n+3)/2 \).

To be more intuitive, figure 1 shows a multi-resolution approximate representation of the time series \( Q \). Each layer represents an approximate representation of the primary resolution, the box represents the mean of a segment, and the number in the box is the start and end point of the segment.

![Figure 1. Multi-resolution approximate representation of the time series.](image)

We can find that one feature point has a one-to-one correspondence with a level approximation. In fact, as a new feature point added, there is a segment broken in the previous level and two new segments created in the current level. Therefore, when saving the multi-resolution representation of the time series, it is not necessary to repeatedly calculate and save all the segmental mean values at
each level of resolution, and only need to save the segments changed in each level. In another word, for each level of resolution, we only need to record three data items and if $A(Q)$ is saved in an array, only $3*n$ data items need to be saved totally.

2.2. Multi-resolution Filtering

We represented the query sequence in multi-resolution representation. For a window sequence, we can be obtained the representation of the window sequence by directly “projecting” the feature points of the query sequence into the window sequence and finding the mean value of the sections of $Q$ that fall within the projected intervals.

The following is a multi-resolution filtering algorithm that matches the similarity between query sequence $Q$ and window sequence $W$.

**Algorithm 1:** multi-resolution filtering (isSimilar)

**Input:** $errCondition$: user-defined similarity threshold, $mrmP$: multiresolution representation of query sequence $Q$, $mrmW$: multi-resolution representation on the sequence in sliding window, $segNodes$: array for storing feature points, $length$: the length of $Q$.

**Output:** $isContinue$: Whether the current sequence can be filtered directly.

1. boolean $isContinue = false$;
2. double $lowerBound = Math.pow((mrmP[0]-mrmW[0]),2) \times length$;
3. double $distance = 0$;
4. int $k=0$; int $i=0$;
5. for($i<=segNodes.length;i++){
6.     $lowerBound = callLowerboundMean(lowerBound,segNodes[i], mrmP, mrmW, k);$;
7.     $distance = Math.sqrt(lowerBound);$;
8.     if($distance > errCondition) {
9.         break;
10.     }
11. } $k+=3$;
12. } $i>=segNodes.length$ {
13.     $isContinue = true$;
14. } $return isContinue$;

We initialize the return value as false at first line. $lowerBound$ is an auxiliary variable used for incrementally calculating the lower bound distance. Actually it is the value of the lower bound distance before extracting a root. The variable named $distance$ holds the lower bound distance of $Q$ and $W$ at each resolution. The filtering procedure repeats in a for-statement with approximations at different scales from $i=0$ to $j=segNodes.length$ (line 8-15). We calculate the lower bound distance between $Q$ and $W$ at the $i^{th}$ resolution (line 9-10), and if the lower bound distance is larger than the similarity threshold, return false directly otherwise calculate at the $i+1^{th}$ resolution (line 11-14) until the most advanced resolution. If the window sequence is not filtered, it is likely to be similar to the query sequence so returns true.

Specifically, $callLowerboundMean()$ is the method for incrementally calculating the lower bound distance. As shown in figure 2, assuming that the $i$-level resolution filtering is performed at present and the current feature point is node the lowerbound in this level is $lowBound(i-1)$. We can get $node.pre$ and $node.next$ from $segNodes[i]$. The segment $S1$ from $node.pre$ to $node.next$ at the $i$-level resolution is divided into two segments $S2, S3$ which are from $node.pre$ to node and node to $node.next$. Calculate the lowerBound of the three segments $S1, S2, S3$ between $Q$ and $W$ respectively, and then lowerBound$(i)=lowBound(i-1)-S1+S2+S3$ at the resolution; After exacting a root of lowerbound, the distance at the $i^{th}$ resolution is available.
2.3. Similarity Search

Based on the multi-resolution representation and multi-resolution filtering method for time series, the algorithm of multi-resolution similarity search is described as follows:

**Algorithm 2:** similarity search

**Input:** double[] $S$, double[] $Q$, double $errCondition$

**Output:** ArrayList $resP$

1. while a new data item $t$ arrived do
2. let $W_i$ be a sliding window with the most recent values containing $t$
3. if(isSimilar(segWeights, $Q$, $W_i$, $errCondition$)){
4. \hspace{1cm} subP.add($W_i$);
5. \hspace{1cm} realDistance = calDistance($Q$, $W_i$);
6. \hspace{1cm} if(realDistance $\leq$ $errCondition$){
7. \hspace{2cm} resP.add($W_i$)
8. \hspace{1cm} }
9. }
10. return $resP$;

Input the query sequence $Q$, time series stream data $S$, similarity threshold $errCondition$, whenever $S$ has new data coming, there is a new sliding window sequence $W_i$. We invoke the filtering algorithm to determine whether the window sequence is filtered (line 3). If it is not filtered, we add it to a candidate set and calculate the real distance between $Q$ and $W_i$ (line 5), finally put it into the result set if it is similarity to the query sequence (line 6-7).

3. Experiments and Analysis

In this section, we compare our multi-resolution search scheme (MRSS) with the MSM method through experiments. We not only focus on the differences in search performance between the two methods, but also on the actual efficiency of searching on different data sets.

3.1. Data Sets and Evaluation Metrics

In order to complete the experiment, we selected some typical opensource datasets from the UCR data archive [10]. These five data sets cover a wide range of application areas and data features.

In our experiments, the goal of our search is to find all subsequences in the time series stream data that matched the given query sequence as quickly as possible. So we use the retrieval time to measure the performance of the search. The retrieval time consists of two parts: 1. the time for multi-resolution representation of the query sequence. 2. the time for matching the query sequence with all window sequences.

3.2. Performance of Similarity Search

We provide the performance of the search method MSM and our MRSS on the above data set. Before the experiment, considering the diversity and complexity of the data sets, we need to define some conditions in advance. Due to the limitations of the MSM method, we have to take the length of the
query sequence to an integer power of two. At the same time, in order to control the experimental variables, we adjust the parameters so that the number of segments is close when our scheme MRSS and the MSM method represent the time series at the highest resolution (we remarked the number as segCounts). We perform the experiment 20 times. For each time we try to measure evaluation metrics above and then take an average. The experimental results are shown in Table 1.

| Dataset       | segCounts: | MSM | MRSS |
|---------------|------------|-----|------|
| ElectricDevices | 16         | 13  | 13   |
| candidate numbers: | 887 | 68  |      |
| represent time: | 1ms | 9ms |      |
| search time: | 753ms | 229ms |      |
| total time: | 754ms | 238ms |      |
| segCounts: | 16 | 18 |      |

| Dataset       | segCounts: | MSM | MRSS |
|---------------|------------|-----|------|
| ECG5000       | 16         | 13  | 13   |
| candidate numbers: | 129 | 47  |      |
| represent time: | 1ms | 6ms |      |
| search time: | 86ms | 43ms |      |
| total time: | 87ms | 49ms |      |
| segCounts: | 32 | 25 |      |

| Dataset       | segCounts: | MSM | MRSS |
|---------------|------------|-----|------|
| FordA_TRAIN  | 32         | 25  | 25   |
| candidate numbers: | 3 | 3 |      |
| represent time: | 1ms | 10ms |      |
| search time: | 809ms | 424ms |      |
| total time: | 810ms | 434ms |      |
| segCounts: | 16 | 17 |      |

| Dataset       | segCounts: | MSM | MRSS |
|---------------|------------|-----|------|
| Worms         | 16         | 13  | 13   |
| candidate numbers: | 9 | 8 |      |
| represent time: | 1ms | 6ms |      |
| search time: | 102ms | 107ms |      |
| total time: | 103ms | 113ms |      |
| segCounts: | 16 | 10 |      |

| Dataset       | segCounts: | MSM | MRSS |
|---------------|------------|-----|------|
| ShapesAll     | 16         | 13  | 13   |
| candidate numbers: | 11 | 11 |      |
| represent time: | 1ms | 9ms |      |
| search time: | 753ms | 229ms |      |
| total time: | 754ms | 238ms |      |

4. Conclusions
In this paper, we proposed a multi-resolution approximate representation of a time series with a binary search tree structure. The multi-resolution representation can be calculated incrementally. The representation of the next level can be calculated from the current level which saves storage space and computational cost. What’s more, we perform the multi-resolution filtering when matching the query sequence and the window sequences. Through this filtering method, the dissimilar sequences can be effectively excluded at low levels, and the calculation is terminated early, thereby improving the retrieval efficiency. Through the above experiments, our multi-resolution search scheme(MRSS) can
provide more accurate and efficient retrieval results than MSM. In future, we plan to use this method as a useful tool for the follow-up streaming time series research.

5. References

[1] Yupeng Hu, Pengjie Ren, Wei Luo, Peng Zhan, Xueqing Li. Multi-resolution Representation with Recurrent Neural Networks Application for Streaming Time Series. Computer Networks, 2019, pp 114-132.

[2] Yupeng Hu, Cun Ji, Ming Jing, and Xueqing Li, “A k-motifs discovery approach for large time series data analysis,” in International Conference on Asia Pacific Web Conference.

[3] Yupeng Hu, Cun Ji, Ming Jing, Yiming Ding, Shuo Kuai and Xueqing Li, “A continuous segmentation algorithm for streaming time series,” in International Conference on Collaborative Computing: Networking, Applications and Worksharing. Springer, 2016, pp. 140–151.

[4] Yupeng Hu, Chong Yang, Cun Ji, Yang Xu, Xueqing Li. Efficient snapshot knn join processing for large data using mapreduce. In Proceedings of IEEE International Conference on Parallel and Distributed Systems (ICPADS), 2016, pp 713–720.

[5] C.Faloutsos et al. Fast Subsequence matching in time-series databases. In Proceedings of ACM SIGMOD, 1994.

[6] Like Gao and X. Sean Wang. Continually evaluating similarity-based pattern queries on a streaming time series. In Proceedings of ACM SIGMOD, 2002.

[7] Like Gao, Zhengrong Yao, and X. Sean Wang. Evaluating continuous nearest neighbour queries for streaming time series via pre-fetching. In Proceedings of international conference on Information and Knowledge Management, 2002.

[8] Yupeng Hu, Peiyuan Guan, Peng Zhan, Yiming Ding, Xueqing Li. A novel segmentation and representation approach for streaming time series. IEEE Access, 2018.

[9] Lian X, Chen L, Yu Jeffrey Xu et al. Similarity match over high speed time-series streams. Proceedings of the 23rd IEEE International Conference on Data Engineering, 2007.

[10] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, “The ucr time series classification archive,” July 2015, www.cs.ucr.edu/_eamonn/time series data/.