Boundary-Based Time Series Sorting

Jun-Kui Li, Yuan-Zhen Wang, and Hai-Bo Li

Abstract—In many applications, it is desirable to sort the data. Most of previous work on sorting are key based, however, there are no apparent keys for the time-series data and therefore the classic sorting algorithms may fail in sorting time-series data. We propose a novel technique, called TS-Sort, to sort time-series sequences in the massive set. The proposed method first extracts the maximum and minimum boundaries of the set, then calculates the distance values between the sequences to the boundaries, and finally sorts the values to determine the relative orders of sequences in the set. For improvement, we propose a partition based version of the algorithm, which puts the sequences into small groups, and sorts the groups to get the final sorted set. Extensive experiments, both on synthetic and real datasets, show that our approach can be used to make the time series set in order, and there is a factor of up to 26.3% accelerating for the improved version of the method.

Index Terms—Boundary comparison, sorting, time series.

1. Introduction

With the growing popularity of time-series data, researches of processing time-series data have received significant attentions in recent years\(^1\). In this work, we consider an important yet interesting problem, i.e., if we have a set of time series, how we sort the sequences in the set?

Applications of sorting time series include but are not limited to the following aspects:

Support for top-k search. The most notable example is the support for top-k search in the similarity search of time series, i.e., searching for the k sequences in the time series database that are most similar to the given query sequence. If we have found a set of candidate sequences, the k sequences with the minimal distance to the query sequence will be opted out. The method for this situation is to sequentially scan the set, calculate the distance, and find out the minimal k sequences. In this sense, if we have the candidate sequences sorted, the k sequences can be easily found. Moreover, we only need to do one sorting, and enjoy the facility of multiple times of search.

Help to build the user-friendly search engine. The requirements from the user may differ in a variety scope, therefore, the search algorithms generally require the setting of many parameters, and a great number of the resulting sequences will be returned to the user if the parameters are inappropriately specified. However, these cases occur very frequently since the user may not have the fully prior understanding of the data under beneath. Now that many sequences have returned, it may still be difficult for the user to find the preferable sequences. However, it will be more user-friendly for the search engine if the utility of order-by is provided to the user. Instead of looking through from a great deal of unsorted sequences by paging-down and paging-up, the user can retrieve the desirable sequences by sorting the results. As another example, the user may wish to display the results in a more human-readable form, and the time series sorting can be adopted to satisfy the requirement.

Sorting has been studied extensively in the area of information retrieval. A variety of classic sorting methods have been proposed\(^2\), such as the QuickSort, HeapSort, ShellSort, BubbleSort, BucketSort, etc. However, it should note that sorting time series differs from these approaches, since the traditional sorting methods are key-based, where the comparisons of entries are performed on a few attributes of the records (usually only one key). While time series is a long sequence of real values, and there are no explicit keys inherited in time series, thus the key-based methods may not be applicable to the time series sorting. To the best of our knowledge, the problem of sorting time series sequences has not been well addressed.

Inspired by the similarity search of time series\(^3\)-\(^5\), where the distance between time series is used to measure the dissimilarity (difference) of the sequences, we propose to adopt the distance between time series for sorting. However, as the distance between two sequences takes no effect on their relative orders in the set, how we sort the sequences with distance? Note that the set of sequences are enclosed by the set boundaries, we can use the distance to the boundary as the basis of sorting, and we call this boundary comparison.
In this work, we propose a partition based method to sort time series. We call our method TS-Sort (time-series sort). Overall, our contributions in this work can be simply summarized as follows:

1) We introduce the boundaries of time series set, and compare the distance of each sequence with the boundary.
2) We propose a method, named TS-Sort, to quickly sort the set of time series.
3) We experimentally evaluate the TS-Sort method both on the synthetic and real datasets. The results of the experiments validate the utilities of TS-Sort.

The rest of the paper is organized as follows. Section 2 provides a background for our work. In Section 3 we present the TS-Sort in details, and show how a set of sequences can be efficiently sorted. The experimental results and analysis on the results are given in Section 4. Finally, we offer our conclusions and discuss the future work in Section 5.

2. Related Work and Definitions

2.1 Related Works

The traditional sorting methods are key-based, and we have mentioned before that the key-based sorting methods may fail in sorting time series. Many works on dimensionality reduction on time series mapped the whole long time series into points in an N dimensional vector space, and treated several dimensions in the transformed space as the keys of the data, which provided a way to represent the time series with the feature vector. Among them are discrete Fourier transform (DFT)\(^6\), discrete wavelet transform (DWT)\(^7\), singular value decomposition (SVD)\(^8\), piecewise aggregate approximation (PAA)\(^9\), and adaptive piecewise constant approximation (APCA)\(^10\). These methods have different features in the time series processing, however, none of them will produce the exact results.

For time series comparison, different distance measures have been proposed\(^1\), and Euclidean distance (more general \(L_p\) norms) have been the most commonly adopted\(^3\). The distance is calculated in a point by point manner and work well when the time series have the same unit of scale. However, when the lengths of two sequences are different or when it is required to match sequences that are locally out of phase, the dynamic time warping (DTW) is usually used instead\(^3\).\(^4\)\(^5\)\(^11\)\(^12\).

In the field of boundaries of time series, Keogh et al. proposed the envelope for the time series sequences \(^4\), where the sequence is enclosed in an Envelope of two sequences \(U\) and \(L\), depicting the Upper and Lower boundaries for the sequence. The work in \(^11\) introduced a notion, named skyline bounding region (SBR), which used a region called SBR to approximate and represented a group of time series data according to their collective shape.

2.2 Problem Definitions

We are now in the position to give a formal description of the problem under considering, the time series sorting.

Intuitively, sorting can be formally depicted with a tuple \(<S, \leq>\), where \(S\) is the set containing all the elements, \(\leq\) is the partial order relation on the elements in \(S\), which defines the relative orders of elements in the sorted set. For any two entries \(O\) and \(O'\) in \(S\), if \(O \leq O'\), the order of \(O\) is smaller than that of \(O'\).

Definition 1. Sorting time series. Given a set of time series \(S\), find a partial order relation \(<S, \leq>\) of the sequences in \(S\), such that for \(\forall T, T' \in S\), \(T \leq T'\) or \(T' \leq T\).

In this work, we propose to calculate the distance of the sequence to the set boundary.

Definition 2. Boundary of time series set. Given a set of time series \(S = T_1, T_2, \ldots, T_c\), where \(T_i = t_{i1}, t_{i2}, \ldots, t_{in}\), for the minimal boundary of time series \(\text{Min}\), \(\text{Min}_k < t_{ik}\), and for the maximum boundary of time series \(\text{Max}\), \(\text{Max}_k > t_{ik}\), \((i = 1, 2, \ldots, c; \ k = 1, 2, \ldots, n)\).

The Max boundary consists of the data points that are maximum in the set, i.e.,

\[
\text{Max}_k = \max\{t_{i1}, t_{i2}, \ldots, t_{ik}\}.
\]

Similarly, the Min boundary is formed by the data points that are minimum in the set, i.e.,

\[
\text{Min}_k = \min\{t_{i1}, t_{i2}, \ldots, t_{ik}\}.
\]

With the introduction of boundary of time series set, the definition of sorting time series changes to Definition 3

Definition 3. Sorting time series with boundary comparison. Given a set of time series \(S\), the boundary sequence \(B\), the distance measure \(d\), find a partial order relation \(<S, \leq>\) of the sequences in \(S\), such that for \(\forall T, T' \in S\),

\[
d(T, B) < d(T', B) \Leftrightarrow T \leq T'.
\]

3. Proposed Methods

3.1 Direct Approach

Since there are two boundaries for the time series set (i.e., Max and Min), the boundary comparison, therefore, can proceed with two types:

A. Comparing with the Max Boundary

The distance between the sequence and the Max boundary of the set is calculated, i.e.,

\[
d(T, \text{Max}) \geq d(T', \text{Max}) \Leftrightarrow T \leq T'.
\]

Note that in (4), the larger the distance from each sequence to the Max boundary is, the smaller the sequence order is.

B. Comparing with the Min Boundary

The distance between the sequence and the Min
boundary of the set is calculated, i.e.
\[ d(T, \text{Min}) \leq d(T', \text{Min}) \Rightarrow T \leq T' \quad (5) \]

From (5), the larger the distance from each sequence to the Min boundary is, the greater the sequence order is.

Table 1 outlines the idea of time series sorting. After extracting the set boundaries, each sequence in the set is compared with the boundary, and sorted with the distance.

Table 1: Outline of time series sorting

<table>
<thead>
<tr>
<th>Algorithm: TS-Sort (Direct)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> S: set of time series</td>
</tr>
<tr>
<td><strong>Output:</strong> The sorted set of time series</td>
</tr>
<tr>
<td><strong>Process:</strong></td>
</tr>
<tr>
<td>1. <strong>Sorted</strong> S ← ∅; // initialize</td>
</tr>
<tr>
<td>2. Calculate the set boundaries with (1) or (2);</td>
</tr>
<tr>
<td>3. for T ∈ S do</td>
</tr>
<tr>
<td>4. Calculate the distance of T to Max or Min;</td>
</tr>
<tr>
<td>5. Sort and get order, ( o_T, ) of T with (4) or (5);</td>
</tr>
<tr>
<td>6. Append( Sorted _S, o_T, T);</td>
</tr>
<tr>
<td>7. end for;</td>
</tr>
<tr>
<td>8. return Sorted _S.</td>
</tr>
</tbody>
</table>

### 3.2 Improved Approach

**Problems with the Direct Approach.** Two problems with the direct approach arise as follows:

1) Only one boundary is considered. Each time only one set boundary Max or Min is used in the calculation.

2) The distance values to the boundary are sorted in the whole set. Note that in the line 5 of the algorithm in Table 1, the values will be sorted within the whole set. However, the distance comparisons can be dramatically reduced if the sequences are partitioned into small groups.

**The Method of Partition.** For improvement, we propose using a partition method to reduce the comparisons of distance. After calculating the distance between the sequences to the boundary, we use (6) to place the sequence into the target sequence group:

\[ g(T) = \frac{K \times d(T, \text{Min})}{d(\text{Max}, \text{Min})} \]  

where \( K+1 \) is the number of groups, we will discuss the selecting of appropriate value for \( K \) in later part. Similarly, we can also use \( d(T, \text{Max}) \) in (6), which corresponds to the case of comparing with the Max boundary in the direct approach.

**Proposition 1.** For \( K > 0 \), \( T \in S \),
\[ 0 \leq g(T) \leq K. \]  

**Proof.** With (1) and (2), we have \( 0 \leq d(T, \text{Min}) \leq d(\text{Max}, \text{Min}) \), thus
\[ 0 \leq \frac{d(T, \text{Min})}{d(\text{Max}, \text{Min})} \leq 1 \]
and for \( K > 0 \),
\[ 0 \leq \frac{K \times d(T, \text{Min})}{d(\text{Max}, \text{Min})} \leq K. \]

**Proposition 2.** \( \forall T, T' \in S, g(T) < g(T') \Rightarrow T \leq T' \).

**Proof.**
\[ g(T) < g(T') \Rightarrow \frac{K \times d(T, \text{Min})}{d(\text{Max}, \text{Min})} < \frac{K \times d(T', \text{Min})}{d(\text{Max}, \text{Min})} \]
\[ \Rightarrow d(T, \text{Min}) < d(T', \text{Min}) \]
\[ \Rightarrow d(T, \text{Min}) < d(T', \text{Min}) \]
\[ \Rightarrow T \leq T'. \]

**Lemma 1.** Equation (6) divides distance space \( D \) into \( K+1 \) groups, \((D_0, D_1, \ldots, D_k), \) where \( D_i = \{d(T, \text{Min}) \mid g(T) = i\} \), \( D_0 < D_1 < \cdots < D_k \).

**Proof.** With (7), we can get the \( K+1 \) groups. We need to prove that for \( \forall i, j(0 \leq i < j \leq K) \), \( D_i < D_j \).

Suppose \( \forall T \in D \) and \( \forall T' \in D_j \). As \( i < j \), i.e. \( g(T) < g(T') \), with Proposition 2, we get \( T \leq T' \), hence \( D_i < D_j \).

The implication of Lemma 1 is exceedingly important. It provides the theoretical basis for partition based sorting on time series. Once one sequence is put into the group, i.e. the \( i \)th \( (0 \leq i \leq K) \) group, Lemma 1 ensures that the order of the sequence in the final set will be greater than the order of those in groups of \( 0, 1, \ldots, i-1 \), and be smaller than the order of those in groups of \( i+1, \ldots, K \). With this in mind, we present the improved version of TS-Sort in Table 2.

Table 2: Improved time series sorting

<table>
<thead>
<tr>
<th>Algorithm: TS-Sort (improved)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> S: set of time series</td>
</tr>
<tr>
<td>K: auxiliary parameter for set partition</td>
</tr>
<tr>
<td><strong>Output:</strong> The sorted set of time series</td>
</tr>
<tr>
<td><strong>Process:</strong></td>
</tr>
<tr>
<td>1. Sorted _S ← ∅; // initialize</td>
</tr>
<tr>
<td>2. Calculate the set boundaries with (1) or (2);</td>
</tr>
<tr>
<td>3. ( d_{\text{bound}} \leftarrow d(\text{Max}, \text{Min}) );</td>
</tr>
<tr>
<td>4. for T ∈ S do</td>
</tr>
<tr>
<td>5. ( k \leftarrow \left[ \frac{K \times d(T, \text{Min})}{d_{\text{bound}}} \right] );</td>
</tr>
<tr>
<td>6. Sort and get order, ( o_T ), of T with (5);</td>
</tr>
<tr>
<td>7. Append( Sorted _S, o_T, T);</td>
</tr>
<tr>
<td>8. end for;</td>
</tr>
<tr>
<td>9. Sorted _S ← ( \cup_i {\text{Sorted } S_i} );</td>
</tr>
<tr>
<td>10. return Sorted _S.</td>
</tr>
</tbody>
</table>

From the improved version algorithm, after extracting the set boundaries Max and Min, we calculate the distance between boundaries Max and Min, \( d_{\text{bound}} \). Each sequence is distributed to the appropriate group with (6), and then the distance comparison, instead of being performed in the whole set, is confined only to the sequences in the same group.

**Time Complexity.** We now analyze the time complexity of the TS-Sort algorithm in Table 2. Let \( c \) denote the
number of sequences in the dataset and \( n \) be the dimensionality of the sequence. First, it takes \( O(cn) \) to compute the set boundaries. The complexity of distance calculation (in lines 3 and 5) depends on the distance measure adopted by the algorithm. For the Euclidean distance (or more general \( L_p \) distance), the complexity is \( O(n) \), while for the DTW distance, the complexity is \( O(n^2) \). During the step of sorting distance, we can use a priority queue to keep trace of the sorted distance values in the group, so it takes \( O(c + \log_2(c)) \) to place a sequence into the group. The final sorted result set is a sequential combination of all the group-wise sets, which can be finished in \( O(1) \). Therefore, total time complexity is:

\[
\begin{align*}
O(cn + c \log_2(c)), & \quad \text{Euclidean}(L_p) \\
O(cn^2 + c \log_2(c)), & \quad \text{DTW}
\end{align*}
\]

(8)

4. Experiment Study

In this section, we examine the methods presented in this work with a comprehensive set of experiments.

4.1 Experiment Study

We conducted experiments on the synthetic and real-life datasets. The datasets used are as follows:

Artificial dataset. The dataset is created using a random time series generator that produces \( n \) time series as \( t_j = t_{j-1} + (-1)^j Z_j \), where \( Z_j (j = 1, 2, \cdots) \) are independent, identically distributed random variables taken in the range of \([0, 1]\). Each sequence is with length of 100, and the base value \( T_{i0} \) is 30. To ensure there are fluctuations in the set, the base value for each sequence, \( T_{i0}(i = 2, 3, \cdots, c) \), increases by 0.05 as the sequence number \( i \) grows.

Synthetic Control Chart. The dataset is from http://kdd.ics.uci.edu/databases/synthetic_control, which contains 600 examples of synthetically generated control charts and can be divided into six classes: normal, cyclic, increasing trend, decreasing trend, upward shift and downward shift.

For these experiments, we used a personal computer and the system configuration is listed in Table 3.

<table>
<thead>
<tr>
<th>Configuration Item</th>
<th>Item Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel Pentium 3-866</td>
</tr>
<tr>
<td>Operating System</td>
<td>GNU/Linux (core: 2.6.1.3)</td>
</tr>
<tr>
<td>RAM</td>
<td>256 megabytes</td>
</tr>
<tr>
<td>Hard disk</td>
<td>Seagate, 40 gigabytes</td>
</tr>
<tr>
<td>Programming Language</td>
<td>ANSI C</td>
</tr>
<tr>
<td>Compiler</td>
<td>GNU gcc 4.1.2 20060928</td>
</tr>
</tbody>
</table>

To demonstrate the results of TS-Sort on the datasets, we adopted a distance graph to describe the distance between the neighboring sequences in the sorted set. The position \( i \) in the distance graph represents the distance between the sequence \( T \) and sequence \( \text{Bound} \), i.e., \( d(T_i, \text{Bound}) \), where \( \text{Bound} \) is the sequence \( \text{Min} \) or \( \text{Max} \).

Our experiments were conducted on the datasets with both Euclidean and DTW distance.

4.2 Results of Time Series Sorting

The Euclidean distance graphs before and after sorting on the artificial dataset are shown in Fig. 1 (a) and Fig.1 (b), respectively. The Euclidean distance graph before sorting is chaotic, and it is difficult to extract the pattern in the sequences directly. However, after we sorting the dataset with TS-Sort, the sequences in the sorted set display in an orderly mode. When the sequences are sorted with the distance to \( \text{Min} \) sequence in ascending order, the distances to the \( \text{Max} \) sequence are approximately sorted in descending order, vice versa.

Fig. 2 presents the results of sorting with DTW distance on the artificial dataset. The similar trend is observed from the results. Compared with the unsorted data, the sorted sets display more in order.
Fig. 2. DTW distance graph of sorting artificial dataset: (a) before sorting and (b) after sorting.

The results of sorting synthetic control chart dataset with Euclidean distance and DTW distance are shown in Fig. 3 and Fig. 4, respectively.

As we introduced before, traditionally the sequences in synthetic control chart dataset were recognized to be divided into six groups by their shapes, and the six sections in the distance graphs before sorting reflect this. However, it can be seen in the distance graphs after sorting that the distance spaces of the sequences can be roughly categorized into three classes. Those sequences with order ranging from 1 to 200 are in the class 1, and those ranging from 201 to 400 are in the class 2, the rest sequences are in the class 3. This indicates that TS-Sort can be used to discover the patterns inherited in the time series data that may not know in advance.

4.3 The Elapsed Time for Sorting

In this part, we present the results on the time during sorting. As we repeated each experiment for 50 times, the results reported here are the average of the elapsed time in the experiment with the same parameters configured. Fig. 5 compares the elapsed time of the Direct approach and the Improved approach on both Artificial dataset and Synthetic Control Chart dataset. When sorting artificial dataset, the number of group number $K$ was set to 8, and the value of $K$ was 6 in sorting synthetic control chart dataset. In each case, we performed four separated experiments, the experiments
labeled in number 1 and 2 used Euclidean distance for the distance measure, and the experiments labeled in number 3 and 4 used DTW distance. The experiments 1 and 3 reported the results of sorting with the Direct approach, and 2 and 4 reported the results of sorting with the Improved approach. For each experiment, we presented three types of time, i.e., time for the boundary calculation (t_bound), time for sorting (t_sort) and time for group division (t_group). Note that there was no group division in the Direct approach, so the t_group was 0.0 in experiments 1 and 3.

![Fig. 5. Elapsed time results.](image)

In Fig. 5, we can see that the Improved approach outperforms the Direct approach on both of the two distance measures, generally above 26.3%. This is due to that sorting with the Improved approach is performed in a relative smaller group, which reduces the overall number of comparisons.

5. Conclusions

In this work, we study the problem of sorting time series. The big challenge on the problem is that time-series sequence has no external keys required by the traditional key-based methods. We calculate the distance to the set boundaries, and sort the values to make the set in order, which is the main idea of our direct approach. To make improvement, we also propose the partition-based TS-Sort method. The extensive experiments show that TS-Sort can be adopted as a useful tool for sorting time series set, and the performance gain is above 26.3% when using the improved version of the method. For future work, we intend to couple the method with other mining methods and explore the possibility for sorting time series in knowledge discovery of time-series data.

References


Jun-Kui Li was born in Jingdezhen, Jiangxi, China, in 1981. He received B.S. degree from Wuhan University of Technology in 2003, now he is a Ph.D. candidate with College of Computer Science and Technology, Huazhong University of Science and Technology (HUST). His research interests include data mining, machine learning.

Yuan-Zhen Wang was born in Wuhan, Hubei, China in 1945. She is now a professor with College of Computer Science and Technology, HUST. Her research interests include modern database technology and data mining.

Hai-Bo Li was born in Wuhan, Hubei, China, in 1975. He is now a Ph.D. candidate in College of Computer Science and Technology, HUST. His research interests include data mining, database theory.