INFORMATICA, 2023, Vol. 34, No. 3, 557–576
© 2023 *Vilnius University*
DOI: https://doi.org/10.15388/23-INFOR519

Multi-Querying: A Subsequence Matching Approach to Support Multiple Queries

Wen LIU[1,2], Mingrui MA[3], Peng WANG[4,*]

[1] *Artificial Intelligence and Smart Mine Engineering Technology Center, Xinjiang Institute of Engineering, Urumqi, China*
[2] *Xinjiang Sunshine Diantong Technology Co., Ltd, Urumqi, China*
[3] *School of Information Science and Engineering, XinJiang University, XinJiang, China*
[4] *School of Computer Science, Fudan University, Shanghai, China*
e-mail: 627952@qq.com, 119345263@qq.com, pengwang5@fudan.edu.cn

Received: August 2022; accepted: May 2023

Abstract. The widespread use of sensors has resulted in an unprecedented amount of time series data. Time series mining has experienced a particular surge of interest, among which, subsequence matching is one of the most primary problem that serves as a foundation for many time series data mining techniques, such as anomaly detection and classification. In literature there exist many works to study this problem. However, in many real applications, it is uneasy for users to accurately and clearly elaborate the query intuition with a single query sequence. Consequently, in this paper, we address this issue by allowing users to submit a small query set, instead of a single query. The multiple queries can embody the query intuition better. In particular, we first propose a novel probability-based representation of the query set. A common segmentation is generated which can approximate the queries well, in which each segment is described by some features. For each feature, the corresponding values of multiple queries are represented as a Gaussian distribution. Then, based on the representation, we design a novel distance function to measure the similarity of one subsequence to the multiple queries. Also, we propose a breadth-first search strategy to find out similar subsequences. We have conducted extensive experiments on both synthetic and real datasets, and the results verify the superiority of our approach.

Key words: multiple queries, subsequence matching, time series mining, fault diagnosis.

1. Introduction

In recent years, the plummet in the cost of sensors and storage devices has resulted in massive time series data being captured and has substantially driven the need for the analysis of time series data. Among various analysis and applications, the problem of subsequence matching is of primordial importance, which serves as the foundation for many other data mining techniques, such as anomaly detection (Boniol and Palpanas, 2020; Boniol *et al.*, 2020; Wang *et al.*, 2022), and classification (Wang *et al.*, 2018; Abanda *et al.*, 2019; Iwana and Uchida, 2020; Boniol *et al.*, 2022).

[*]Corresponding author.

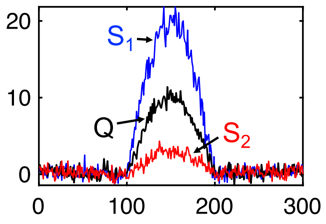


Fig. 1. EOG pattern.

Specifically, given a long time series X, for any query series Q, the subsequence matching problem finds the K number of subsequences from X most similar to Q (top-K query), or finds subsequences whose distance falls within the threshold ε (range query). In the last two decades, plenty of works have been proposed for this problem. Most existing works find results based on the strict distance, like Euclidean distance and Dynamic Time Warping. Among them are scanning-based approaches (Li *et al.*, 2017; Rakthanmanon *et al.*, 2012) and index-based ones (Linardi and Palpanas, 2018; Wu *et al.*, 2019).

Another type of works define the query in a more flexible way. Query-by-Sketch (Muthumanickam *et al.*, 2016) and SpADe (Chen *et al.*, 2007) approximate the query with a sequence of line segments, and find subsequences that can be approximated in a similar way.

Nevertheless, in many real applications, users are unable to accurately and clearly elaborate the query intuition with a *single* query sequence. Specifically, users may have different strictness requirements for different parts of the query. We illustrate it with the following two examples.

Case study 1. In the field of wind power generation, Extreme Operating Gust (EOG) (Hu *et al.*, 2018) is a typical gust pattern which is a phenomenon of dramatic changes of wind speed in a short period. Early detection of EOG can prevent the damage to the turbine. A typical pattern of EOG, as Q and S_1 in Fig 1, has three physical phases, where its corresponding shape contains a slight decrease (1–100), followed by a steep rise and a steep drop (101–200), and a rise back to the original value (200–300). Users usually emphasize the steep increase/decrease in the second part, which means S_1 is more preferred compared to S_2 in Fig 1. However, if the analyst submits query Q, S_2 is more similar to Q under either ED or DTW distance.

Case study 2. During the high-speed train's work time, the sensor will continuously collect vibration data for monitoring. When the train passes by some source of interference, the value of sensors will increase sharply, and return to a normal value after some time, as Q, S_1 and S_2 in Fig 2. However, if Q is issued as a query, subsequence S_3 is more similar to it, which is an unexpected result. By combining Q, S_1 and S_2 together, we can learn that the pattern occurrences may have variable durations and distinct amplitudes. The most strict constraint is that the pattern should include an almost upright rise and an almost upright fall.

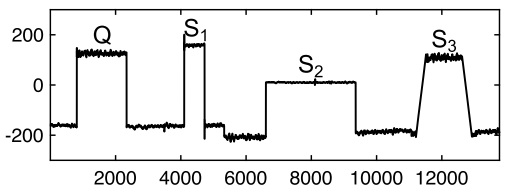


Fig. 2. Signal interference pattern.

The above examples clearly demonstrate the limitation of the single query mechanism. Although a single query can express the shape the user is interested with, it is not enough to express the extent of time shifting and amplitude scaling, as well as the error range.

To solve this problem, in this paper, we propose a multiple query approach. Compared to a single query, submitting a small number of queries together can express the query intuition more accurately. Consider the example in Fig. 2: if users take Q, S_1 and S_2 as the query set, it indicates that the user can show more tolerance in realtion to the subsequence length and the value range, but less in relation to the slope of the increasing and decreasing part. Moreover, submitting multiple queries is also a natural solution in the real-world applications. For example, in the above case of train monitoring, analysts hope to find out all interfered subsequences, and then correct them. The analyst will go through a small part of the long sequence. Once coming across a few interfered subsequences, he/she can submit them together in order to find more instances.

We first propose a *probability-based representation* of the multiple queries. Then, we design a novel distance function to measure the similarity of one subsequence to the multiple queries. In the end, a breadth-first search algorithm is proposed to find out the desired subsequences. To the best of our knowledge, this is the first work to study how to express the query intuition.

Our contributions can be summarized as follows:

- We analyse the problem of expressing the query intuition, and propose a multi-query mechanism to solve this problem.
- We present a probability-based representation of the multiple queries, as well as the corresponding distance between it and any subsequence.
- We present a breadth-first search algorithm to efficiently search for similar subsequences.
- We conduct extensive experiments on both synthetic and real datasets to verify the effectiveness and efficiency of our approach.

The rest of the paper is organized as follows. The related works are reviewed in Section 2. Section 3 introduces definitions and notations. In Section 4 and 5, we introduce our approach in detail. Section 6 presents an experimental study of our approach using synthetic and real datasets, and we offer conclusions in Section 7.

2. Related Work

In last two decades, the problem of subsequence matching has been extensively studied. Existing approaches can be classified into two groups:

Fixed Length Queries: Traditionally, to find out similar subsequences, two representative distance measures are adopted, Euclidean distance (ED) and Dynamic Time Warping (DTW). ED computes the similarity by a one-to-one mapping while DTW allows disalignment and thus supports time shifting. UCR Suite (Rakthanmanon *et al.*, 2012) is a well-known approach that supports both ED and DTW for subsequence matching and proposes cascading lower bounds for DTW to accelerate search speed. FAST (Li *et al.*, 2017) is based on UCR Suite, and further proposes some lower bounds for the sake of efficiency. Both UCR Suite and FAST have to scan the whole time series to conduct distance computation. EBMS (Papapetrou *et al.*, 2011), however, reduces the subsequence matching problem to the vector matching problem, and identifies the candidates of matches by the search of nearest neighbours in the vector space. Also, some index-based approaches have been proposed for similarity search. Most of them build indexes based on summarizations of the data series (e.g. Piecewise Aggregate Approximation (PAA) (Keogh *et al.*, 2001), or Symbolic Aggregate approXimation (SAX) (Shieh and Keogh, 2008)). Coconut (Kondylakis *et al.*, 2018) overcomes the limitation that existing summarizations cannot be sorted while keeping similar data series close to each other and proposes to organize data series based on a z-order curve. To further reduce the index creation time, adaptive indexing techniques have been proposed to iteratively refine the initial coarse index, such as ADS (Zoumpatianos *et al.*, 2016).

Variable Length Queries: For variable length queries, SpADe (Chen *et al.*, 2007) proposes a continuous distance calculation approach, which is not sensitive to shifting and scaling in both temporal and amplitude dimensions. It scans data series to get local patterns, and dynamically finds the shortest path among all local patterns to be the distance between two sequences. Query-by-Sketch (Muthumanickam *et al.*, 2016) proposes an interactive approach to explore user-sketched patterns. It extracts shape grammar, a combination of basic elementary shapes, from the sketched series, and then applies a symbolic approximation based on regular expressions. To better satisfy the user, Eravci and Ferhatosmanoglu (2013) attempts to improve the search results by incorporating diversity in the results for relevance feedback. Relatively speaking, indexing for variable length queries is more intractable. KV-match$_{\text{DP}}$ (Wu *et al.*, 2019) utilizes multiple varied-length indexes to support normalized subsequence matching under either ED or DTW distance. ULISSE (Linardi and Palpanas, 2018), by comparison, uses a single index to answer similarity search queries of variable length. It organizes the series and their summaries in a hierarchical tree structure called the *ULISSE* index.

In summary, up to now, no existing work has attempted to express the query intuition via the multi-query mechanism.

3. Preliminaries

In this section, we begin by introducing all the necessary definitions and notations, followed by a formal problem statement.

3.1. *Definition*

In this work, we are dealing with time series. A time series $X = (x_1, x_2, \dots, x_N)$ is an ordered sequence of real-valued numbers, where $N = |X|$ is the length of X. A subsequence S, S' or $X[i, j] = (x_i, x_{i+1}, \dots, x_j)$ $(1 \leqslant i \leqslant j \leqslant n)$ denotes the continuous sequence of length $j - i + 1$ starting from the i-th position in X. Note that the subsequence is itself a time series.

Given a time series X, a query sequence Q, and a distance function D, the problem of subsequence matching is to find out the top-K subsequences from X, denoted as $\mathbb{R} = \{S_1, S_2, \dots, S_K\}$, which are most similar to Q.

The two representative distance measures are *Euclidean Distance* (ED) and *Dynamic Time Warping* (DTW). Formally, given two length-L sequences, S and S', their ED and DTW distance can be computed as the following:

Definition 1. Euclidean Distance: $ED(S, S') = \sqrt{\sum_{i=1}^{L}(s_i - s'_i)^2}$, where s_i and s'_i is the value at i-th $(1 \leqslant i \leqslant L)$ time stamp of S or S' respectively.

Definition 2. Dynamic Time Warping:

$$DTW(\langle\rangle, \langle\rangle) = 0; \qquad DTW(S, \langle\rangle) = DTW(\langle\rangle, S') = \infty;$$

$$DTW(S, S') = \sqrt{(s_1 - s'_1)^2 + \min \begin{cases} DTW(suf(S), suf(S')), \\ DTW(S, suf(S')), \\ DTW(suf(S), S'), \end{cases}}$$

where $\langle\rangle$ indicates empty series and $suf(S) = (s_2, \dots, s_L)$ is a suffix subsequence of S.

In this paper, instead of processing one single query, we attempt to find out subsequences similar to multiple queries. That is, given a set of queries, $\mathbb{Q} = \{Q_1, Q_2, \dots, Q_N\}$, our objective is to find out the top-K subsequences similar to queries in $\mathbb{Q}$, denoted as $\mathbb{R}$.

Since each query sequence varies in length, we do not impose a constraint on the length of subsequences in $\mathbb{R}$. In this way, we find out variable-length subsequences answering multiple queries, which is worthy of wide use in real time series applications.

4. Query Representation and Distance Definition

In this section, we first present the probability-based representation of the query set, and then propose a distance definition based on the representation.

4.1. *Query Representation*

In this paper, instead of processing the queries in $\mathbb{Q}$ independently, we first represent them by a unified formation, which is a multi-dimensional probability distribution. Then we find target subsequences from X based on the representation.

In many real world applications, the meaningful query sequence can be approximately represented as a sequence of line segments. Recall the EOG pattern in Fig. 1. The query sequence Q can be approximated with 4 line segments. These line segments capture the most representative characteristics in Q.

In response, we propose a two-step approach to represent the bundle of queries together. In the first step, we represent each single query Q_i in $\mathbb{Q}$ individually by a traditional segmentation in which each segment is described by some features. Then, in the second step, we represent each feature as a Gaussian distribution over the values from multiple queries.

4.1.1. *Step One: Represent Each Single Query*

In step one, we perform a traditional segmentation. We use a bottom-up approach to convert the query $Q_i = (q_1, q_2, \dots, q_f)$ into a piecewise linear representation, where q_f is the segment of single query Q_i $(1 \leqslant f \leqslant |Q_i|)$. Initially, we approximate Q_i with $\lfloor \frac{|Q_i|}{2} \rfloor$ line segments. The j-th line, H_j, connects q_{2j-1} and q_{2j}. Next, we iteratively merge the neighbouring lines. In each iteration, we merge the two neighbouring segments into one new line segment that has the minimal approximation error. The merging process repeats until we have m (a pre-set parameter) number of line segments. For each Q_i, we obtain its segmentation, denoted as $(Q_i^1, Q_i^2, \dots, Q_i^m)$ and its linear representation, denoted as $(H_i^1, H_i^2, \dots, H_i^m)$.

For each line segment H_i^j $(1 \leqslant i \leqslant N$ and $1 \leqslant j \leqslant m)$, we represent it as a 4-dimension vector, $f_i^j = (l_i^j, \theta_i^j, v_i^j, \varepsilon_i^j)$, which corresponds to the length, slope, the value of the starting point and MSE error of H_i^j, respectively. As a result, the query sequence Q_i is represented by a $4m$-dimension vector, $F_i = (f_i^1, f_i^2, \dots, f_i^m)$.

4.1.2. *Step Two: Represent Multiple Queries*

After obtaining F_i's $(1 \leqslant i \leqslant N)$, we can generate the uniform representation of query set $\mathbb{Q}$, which is a multi-dimensional probability distribution. We first present the formal distribution, and then give our approach to generate the specific distribution for $\mathbb{Q}$.

Specifically, given query set $\mathbb{Q}$, its representation, denoted as $P_{\mathbb{Q}}$, consists of $4m$ number of individual Gaussian distributions, each of which corresponds to a feature in f_i^j. For each feature, we produce a Gaussian distribution to capture latent semantics, which is determined by two parameters: the mean value and the standard deviation. The former encodes the ideal value of the feature, while the latter provides an elastic range.

Formally, we denote the representation of $\mathbb{Q}$ as $P_{\mathbb{Q}} = (P^1, P^2, \dots, P^m)$, where $P^j = (p_l^j, p_\theta^j, p_v^j, p_\varepsilon^j)$ corresponds to the j-th line segments $(H_1^j, H_2^j, \dots, H_N^j)$. Take the slope feature as an example, that is, p_θ^j is the Gaussian density function of the slope of the j-th segment, denoted as

$$p_\theta^j(x) = \frac{1}{\sqrt{2\pi}\sigma_\theta^j} \exp\left(-\frac{(x-\mu_\theta^j)^2}{2(\sigma_\theta^j)^2}\right), \tag{1}$$

where μ_θ^j (or σ_θ^j) is the mean value (or standard deviation) of the slope values of $(H_1^j, H_2^j, \ldots, H_N^j)$. Specifically, $\mu_\theta^j = \frac{\sum_{i=1}^N \theta_i^j}{N}$, $\sigma_\theta^j = \sqrt{\frac{1}{N}\sum_{i=1}^N(\theta_i^j - \mu_\theta^j)^2}$. The mean value μ_θ^i describes the slope the user prefers, and the standard deviation σ_θ^i represents how strictly the user stresses on this feature. Apparently, the smaller the value of σ_θ^i is, the stricter the user's requirement.

Now we introduce our approach to generate $P_{\mathbb{Q}}$ for the query set $\mathbb{Q}$. We first get the representations $(l_i^j, \theta_i^j, v_i^j, \varepsilon_i^j)$ that correspond to the features of the j-th line segment in Q_i. To get the specific Gaussian distribution $P^j = (p_l^j, p_\theta^j, p_v^j, p_\varepsilon^j)$, we directly compute the mean value and the standard deviation of the feature values of $(H_1^j, H_2^j, \ldots, H_N^j)$. Then, $P_{\mathbb{Q}} = (P^1, P^2, \ldots, P^m)$.

4.2. *Distance Definition*

Given the fact that the query representation consists of $4m$ number of Gaussian distributions rather than a sequence, the existing distance measures, like ED and DTW, are inapplicable. In this paper, we propose a novel distance function $D(S, \mathbb{Q})$ based on the probability distribution.

Formally, to define the distance between subsequence S and the query representation $P_{\mathbb{Q}}$, we first approximate S with m line segments. We indicate the segmentation as $seg = (S^1, S^2, \ldots, S^m)$, where S^j $(1 \leqslant j \leqslant m)$ denotes the j-th segment of S. Note that the segment here infers *subsequence* rather than *line segment*. We extract four features, the length l^j, the slope θ^j, the value of the starting point v^j, and the MSE error ε^j from the linear representation of S^j. For ease of presentation, we represent all the j-th segments in $\mathbb{Q}$ as $\mathbb{Q}^j$. That is, $\mathbb{Q}^j = (Q_1^j, Q_2^j, \ldots, Q_N^j)$. Then the distance between S^j and $\mathbb{Q}^j$ is

$$dist(S^j, \mathbb{Q}^j) = -\log\left(p_l^j(l^j)p_\theta^j(\theta^j)p_v^j(v^j)p_\varepsilon^j(\varepsilon^j)\right), \tag{2}$$

dist$(S^j, \mathbb{Q}^j)$ is the negative logarithm of the probability. The smaller the value, the more similar S^j and $\mathbb{Q}^j$ are. Accordingly, under segmentation seg, the distance between S and the query set $\mathbb{Q}$ can be computed as

$$D(S, \mathbb{Q}, seg) = \sum_{j=1}^{m} dist(S^j, \mathbb{Q}^j). \tag{3}$$

Since S can be segmented by different segmentations, the value of $D(S, \mathbb{Q}, seg)$ may be different. In this paper, we define the distance between S and $\mathbb{Q}$ as the minimal one among all possible segmentations, that is,

$$D(S, \mathbb{Q}) = \min_{seg} D(S, \mathbb{Q}, seg). \tag{4}$$

5. Query Processing Approach

In this section, we introduce the search process. Obviously, it is exhaustive to find out the best segmentation of all subsequences in the time series X. In response, we divide the search process into two phases:

1. Candidate generation. Given the submitted query set $\mathbb{Q}$, we utilize a Breadth-First Search (BFS) strategy to find at most n_c number of candidates from X, denoted as CS.
2. Post-processing. All subsequences in CS will be verified and re-ordered by computing its actual distance. Moreover, we dismiss the trivial match subsequences.

5.1. *BFS-Based Search Process*

We first introduce our search strategy to generate the candidate set CS with size not exceeding the parameter n_c. We utilize an iterative approach, and generate the candidates segment-by-segment. In the first round, we generate at most n_c number of candidates with only one segment. The candidate set is denoted as $CS_1 = \{cs_1, cs_2, \ldots, cs_{n_c}\}$. Each candidate, cs_i, is a triple $\langle s_i, e_i, d_i \rangle$, in which s_i is its starting point and e_i is its ending point. So cs_i corresponds to the subsequence $X[s_i, e_i]$. The third element d_i is distance $dist(cs_1, \mathbb{Q}^1)$. All candidates in CS_1 are ordered based on the values of d_i ascendingly. In other words, CS_1 contains n_c number of subsequences with smallest distance with $\mathbb{Q}^1$. We discuss how to select top-n_c candidates in the next section.

In the second round, we obtain candidate set CS_2 by extending the candidates in CS_1 with the second segment. Specifically, given any candidate subsequence $cs = \langle s, e, d \rangle$ in CS_1, if we want to extend cs to cs' with a length-L segment, the new candidate $cs' = \langle s, e', d' \rangle$ contains two segments: one corresponds to $X[s, e]$ and the other to $X[e + 1, e + L]$. Note that the new starting point s keeps unchanged, and the new ending point e' changes to $e+L$. Also, the new distance d' is updated to $d+dist(X[e+1, e'], \mathbb{Q}^2)$. Since from each candidate cs in CS_1, we can extend it to multiple candidates by concatenating $X[s, e]$ with variable length segments, we generate all possible candidates, compute the distance and add top-n_c candidates into CS_2.

After m rounds, we obtain the candidate set CS_m, which is the final candidate set CS. Now each candidate in CS consists of m segments.

5.2. *Candidate Generation*

Now we introduce our approach to generate possible candidates in each round.

In the first round, we enumerate all subsequences $X[s, e]$ from all possible starting points, that is, we try all s's within $[1, n]$. To avoid the low-quality candidates, we only select subsequences with length satisfying the 3σ standard. Formally, given any starting point s, the ending point e must satisfy $e \in [\mu_l^1 - 3\sigma_l^1, \mu_l^1 + 3\sigma_l^1]$. For each candidate subsequence $X[s, e]$, we compute the optimal linear approximation, $y = \theta \cdot x + b$, as well as the MSE error ε as follows,

$$\begin{cases} \theta = \dfrac{12\sum ix - 6(l+1)\sum x}{l(l+1)(l-1)}, \\ b = \dfrac{6\sum ix - 2(2l+1)\sum x}{l(1-l)}, \\ \varepsilon = \sum x^2 + \theta^2 \sum i^2 + lb^2 - 2\theta \sum ix - 2b \sum x + 2\theta b \sum i, \end{cases} \tag{5}$$

where $l = e - s + 1$ is the length of $X[s, e]$. After that, we compute $\mathit{dist}(X[s, e], \mathbb{Q}^1)$. Also, if only any other feature of a candidate (θ, v or ε) violates the 3σ standard, we ignore this candidate. During enumerating different candidates, we maintain CS_1 as a priority queue to keep top-n_c candidates.

In the j-th round ($2 \leqslant j \leqslant m$), we generate candidates by extending previous ones in CS_{j-1}. For each candidate $cs = \langle s, e, d\rangle$ in CS_{j-1}, we try all possible segments next to $X[s, e]$ whose length falls within $[\mu_l^j - 3\sigma_l^j, \mu_l^j + 3\sigma_l^j]$. Similar with the first round, we also dismiss the candidates violating the 3σ standard. When extending candidate $cs = \langle s, e, d\rangle$ to $cs' = \langle s, e', d'\rangle$, except the new ending point e', we also update the distance as $d' = d + \mathit{dist}(X[e+1, e'], \mathbb{Q}^j)$.

5.3. *Post-Processing*

Note that the subsequences in CS are not approximated optimally. Here, an additional refinement step has to be performed, where subsequences are verified and re-ordered using the optimal segmentation. Specifically, we fetch the subsequences in CS, approximate each subsequence cs with m line segments via a dynamic programming algorithm, and thus get the actual distance $D(cs, \mathbb{Q})$ under the new segmentation.

The objective of the segmentation is to minimize the distance between cs and $\mathbb{Q}$. We search the optimal segmentation from left to right sequentially on cs. We define $E(i, j)$ ($1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant |cs|$) to be the minimal distance between the prefix of cs (i.e. $cs = X[1, j]$) and the prefix of $\mathbb{Q}$ with i segments (i.e. $[\mathbb{Q}^1, \mathbb{Q}^2, \dots, \mathbb{Q}^i]$). We begin by initializing $E(1, j)$ to be $\mathit{dist}(X[1, j], \mathbb{Q}^1)$. When computing $E(i, j)$ ($2 \leqslant i \leqslant m$), we consider all the possible segmentations of $X[1, k]$ ($i \leqslant k \leqslant j$) with $i - 1$ segments, compare the sum of $E(i-1, k)$ and $\mathit{dist}(X[k, j], \mathbb{Q}^k)$, and define $E(i, j)$ to be the minimal one. Formally, the dynamic programming equation is presented as the following

$$E(i, j) = \begin{cases} \mathit{dist}(X[1, j], \mathbb{Q}^1), & i = 1, \\ +\infty, & i > j, \\ \min\limits_{i \leqslant k \leqslant j} \big(E(i-1, k) + \mathit{dist}\big(X[k, j], \mathbb{Q}^k\big)\big), & \text{otherwise.} \end{cases} \tag{6}$$

We re-compute the distance between each subsequence in CS and $\mathbb{Q}$. Before re-ordering, we have to dismiss the trivial match subsequences since candidates can overlap with each other. Formally, given two candidates $cs = X[s, e]$ and $cs' = X[s', e']$, if their overlapping ratio, $\frac{cs \cap cs'}{cs \cup cs'}$, exceeds $\min(0.8, \frac{\min(|Q|)}{\max(|Q|)})$, where the latter indicates the ratio between the minimum length and the maximum length of queries $\mathbb{Q}$, we dismiss the subsequence with the larger distance.

Afterwards, we simply sort the remaining subsequences in CS according to $D(cs, \mathbb{Q})$ and select the K smallest as the final results $\mathbb{R}$.

5.4. *Optimization*

In this section, we propose two optimization strategies to further accelerate the search process.

5.4.1. *Basic Aggregates Based Linear Representation Computation*

In the search process, for each candidate, we adopt linear regression to find out the best line segment $y = \theta \cdot x + b$ in a least square sense using Eq. (5). It is noteworthy that θ, b, and ε can be computed by some combination of three basic aggregates: $\sum x$, $\sum x^2$ and $\sum ix$, with cost $O(1)$, as proposed in Wasay *et al.* (2017). As long as we maintain these three in-memory arrays to store these basic aggregates of time series X, linear regression can be conducted in $O(1)$ time for any subsequence in X.

However, to process extremely long sequences, the storage overheads are unaffordable. As a consequence, we propose to split the time series into several blocks and conduct subsequence matching within each block sequentially and summarize the results in the end. Obviously, this approach reduces memory consumption while being accurate and efficient.

5.4.2. *Adjusting the Searching Order*

Up to now, we have generated the candidates segment-by-segment sequentially. However, different standard deviation of the feature values of different segments in $\mathbb{Q}$ will result in different search space. For example, in $\mathbb{Q}$, one segment has the standard deviation of length $\sigma_l = 5$, while in other segment, $\sigma_l = 10$. According to the 3σ standard, the former has $5 \cdot 3 \cdot 2 + 1 = 31$ candidates, while the latter has 61 candidates. Obviously, we should first search based on the latter. Specifically, we perform the search process in an optimized order, where we first consider the segment whose standard deviation of the value of length feature is the smallest, and then consider its neighbouring segments. Suppose there are 3 sets of line segments in $\mathbb{Q}$, i.e. $m = 3$, their standard deviations of the value of length feature are 2, 3, 1 respectively, that is, $\sigma_l^1 = 2$, $\sigma_l^2 = 3$, $\sigma_l^3 = 1$. Then, in the search process, we generate candidates from right to left sequentially.

5.5. *Complexity Analysis*

The overall process of our approach can be divided into two phases: query representation and query processing.

Before the two phases, we have to scan the whole time series X once to store the basic aggregates. Its time cost is $O(n)$, where n is the length of X.

Then, in the first phase, we scan all the queries and perform traditional piecewise linear approximations. Thanks to the basic aggregates, the time cost of conducting linear regression is negligible. This phase is then $O(N|Q|^2)$ in time complexity, where N is the number of queries. For ease of presentation, although multiple queries can vary in length, we denote $|Q|$ as the length of the queries.

In the second phase, we first assume $w_i = 6\sigma_l^i + 1$, where σ_l^i is the standard deviation of the length values of $(H_1^i, H_2^i, \dots, H_N^i)$. Suppose we generate candidates from left to right sequentially; when considering the first segments, it requires $O(w_1 n \log(n_c))$ time to maintain the candidate set. For the following i-th segments, the time complexity is $O(w_i n_c \log(n_c))$. In the post-processing process, we have to verify and compute the actual distance between $\mathbb{Q}$ and every subsequence cs in the candidate set. It takes $O(n_c m |cs|^2)$ time to conduct the dynamic programming algorithm for the n_c candidates, where m is the number of line segments. Moreover, the re-ordering process is at most $O(n_c \log n_c)$ in time complexity.

Since w_i is a constant, n_c is proportional to n, and $|cs|$ and $|Q|$ is much smaller than n, we can infer that the time complexity of our approach is $O(n \log(n_c))$ theoretically.

6. Experiments

In this section, we conduct extensive experiments to verify the effectiveness and efficiency of our approach. All experiments are run on a PC with Intel Core i7-7700K CPU (8 cores @ 4.2 GHz) and 16 GB RAM.

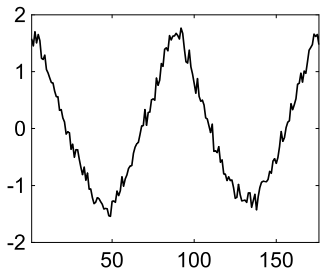
6.1. *Datasets*

6.1.1. *Synthetic Datasets*

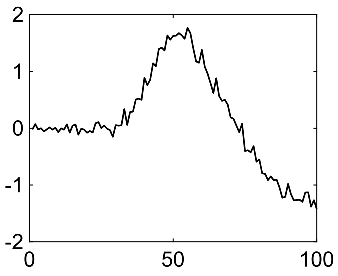
We generate the synthetic sequence as follows. First, we generate a long random walk sequence T. Then we embed in T some meaningful pattern instances, in which some are taken as queries, and others as target results.

We generate two types of patterns, W-wave from the Adiac dataset in UCR archive (Dau *et al.*, 2019), and backward wave, common in stock series. For each pattern, we first generate a seed instance and add some noise following a Gaussian distribution $N(0, 0.05)$, as shown in Fig. 3. Then we modify them to generate more instances. Specifically, we adopt three types of variations, length, amplitude, and shape as follows.

- Length: It is obvious that the W-wave in Fig. 3(a) can be split into four segments. For the middle two segments, we change the segment length with a scaling factor λ by inserting or deleting some data points. In other words, for a length-l segment, the length of the new segment is $l \cdot \lambda$.
- Amplitude: Similar to the case of length scaling, we increase or decrease the amplitude of the middle two segments in the W-wave. We still use a factor λ to control the extent. Specifically, every value in new segment changes to $v \cdot \lambda$, where v is the original value.



(a) W-wave



(b) Backward wave

Fig. 3. Two fundamental patterns.

Table 1
Influence of variations.

	l	θ	v	ε
Length	√			√
Amplitude		√	√	√
Shape	√		√	√

- Shape: For the backward wave shown in Fig. 3(b), we change the global shape of the pattern by modifying the last two segments on both length and amplitude. To be more specific, we change the segment length from l to $l \cdot \lambda$, and the data values from v to $v \cdot \lambda$.

Note that the three types of variations will influence different features in the line segments. We list them respectively in Table 1.

For each type of variation, we test our approach under different extents. Take the length variation as an example. To generate a dataset, we first set a parameter r, which determines the length variation range. Specifically, given r, we can only set the length scaling factor λ within the range $[1/(1+r), 1+r]$. Obviously, the larger the value of r, the larger the length scaling extent.

For each variation, given the fixed r, we pick out 50 values from the range $[1/(1 + r), 1 + r]$ as the value of λ. Then we generate 50 corresponding pattern instances, 40 of which are randomly planted into the random walk long time series. The rest 10 instances form the query set $\mathbb{Q}$. A summary of the synthetic datasets is presented in Table 2.

Table 2
A summary of the synthetic datasets.

Dataset	Pattern	Variation	Length
1	W-wave	Length	0.7 million
2	W-wave	Amplitude	0.7 million
3	Backward wave	Shape	0.4 million

6.1.2. *Real Datasets*

The real dataset is the train monitoring dataset, which is the time series collected by the vibration sensor. Its total length is 15 million. There exist more than 100 interference subsequences that vary in length and amplitude. The length of these subsequences is within the range of 200 to 2500. Consequently, we maintain a query set of size 15 whose lengths almost distribute uniformly between 200 and 2500. The rest ones are leaved as the target results.

6.2. *Counterpart Approaches*

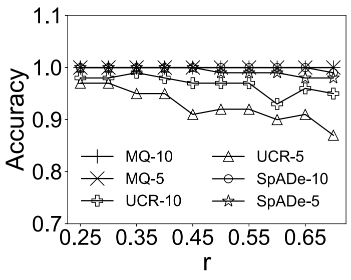
Note that it is difficult to find reasonable baselines to compare our approach to because the existing methods are only devised for the case of a single query. Since no approach can deal with multiple queries as a whole, we choose two representative subsequence matching algorithms, UCR Suite (Rakthanmanon *et al.*, 2012) and SpADe (Chen *et al.*, 2007), and then enable them to handle the problem of multiple queries. UCR Suite finds the best normalized subsequences and supports both Euclidean distance and Dynamic Time Warping (UCR-ED and UCR-DTW for short). SpADe finds the shortest path within several local patterns, and is able to handle shifting and scaling both in temporal and amplitude dimensions.

To make UCR-Suite (or SpADe) support multiple queries, we first find out top-K similar subsequences for each query based on UCR-Suite (or SpADe). We then sort these $K \cdot N$ number of subsequences in the ascending order of their distances (normalized by the subsequence length) and pick out the top-K ones excluding any trivial results as final. For UCR-Suite, we utilize both ED and DTW as the distance, denoted as UCR-ED and UCR-DTW, respectively.

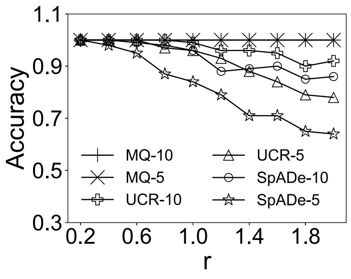
Let N be the number of queries in $\mathbb{Q}$, we denote our approach and the other three competitors as MQ-N, UCR-ED-N, UCR-DTW-N and SpADe-N, respectively. Note that the three rivals have to scan the time series for N times. For fairness, we do not count I/O time when comparing efficiency.

6.3. *Results on Synthetic Datasets*

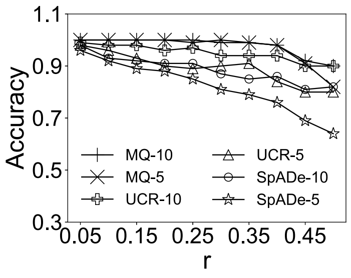
In the first experiment, we compare our approach MQ with UCR-ED, UCR-DTW, and SpADe on synthetic datasets. Both accuracy and efficiency are tested. To compare these approaches extensively, we vary the parameter r for all three variations, length, amplitude and shape. Specifically, for the length variation, we set the maximal scaling factor r from



(a) Length scaling



(b) Amplitude scaling



(c) Shape scaling

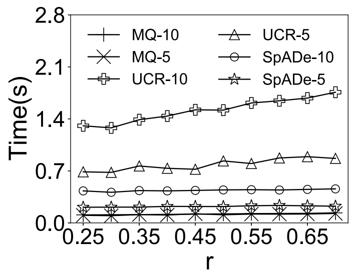
Fig. 4. Accuracy comparisons under different variations.

0.25 to 0.7 with step 0.05. For the amplitude variation, we varied r in a range of 0.2 to 2 with step 0.2. For the shape variation, we changed r in a range of 0.05 to 0.5 with step 0.05.

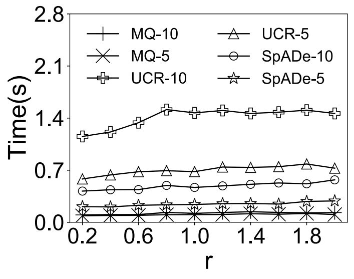
We set the number of queries, N, as 5 and 10, respectively. The experimental results are then indicated by MQ-5, MQ-10, UCR-DTW-5, UCR-DTW-10, SpADe-5, and SpADe-10.[1] For MQ, we set the only parameter, the size of the candidate set n_c, to be $0.05 \cdot n$, where n is the length of the time series X.

In each set of experiments, we attempt to find out the top-40 subsequences in the time series. The accuracy is the ratio between the number of *correct* subsequences and 40. Subsequence S is correct, if the overlapping ratio between S and certain planted instance exceeds the tolerance parameter, $\epsilon = \min\left(0.8, \frac{\min(|Q|)}{\max(|Q|)}\right)$.

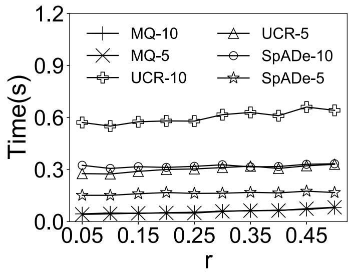
[1]UCR-ED is omitted for its consistent inferiority to UCR-DTW.



(a) Length scaling



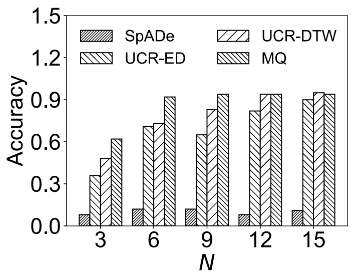
(b) Amplitude scaling



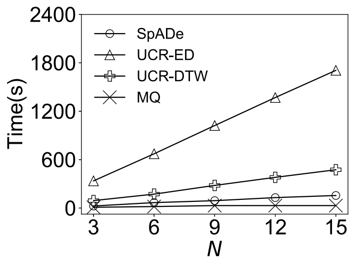
(c) Shape scaling

Fig. 5. Efficiency comparisons under different variations.

The results are shown in Fig. 4 and Fig. 5, respectively. It can be seen that MQ outperforms UCR-DTW and SpADe in both accuracy and efficiency under all variations. The reason is that MQ summarizes common characteristics in multiple queries while the other approaches are only able to find out the subsequences that are similar to certain given query. As a consequence, when the number of query sequences N increases, UCR-DTW and SpADe yield better results. Nevertheless, UCR-DTW-10 (or SpADe-10) is twice as slow as UCR-DTW-5 (or SpADe-5) on average, which means with more query sequences provided, UCR-DTW and SpADe can find out more satisfying subsequences, but at the cost of efficiency. Instead, MQ captures latent semantics and thus demonstrates its superiority in both accuracy and efficiency under different variations. The only exception is in Fig. 4(c), where the accuracy of MQ sharply decreases when $r = 0.4$, which is mainly because the undue scaling in shape will result in ambiguity of what users really want.



(a) Accuracy vs. N



(b) Efficiency vs. N

Fig. 6. Comparison on real dataset.

6.4. *Results on Real Datasets*

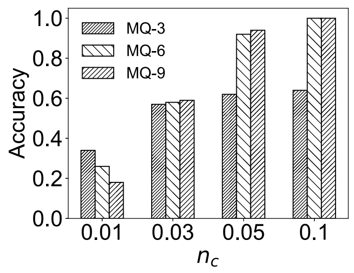
In this experiment, we compare our approach with other ones as the number of query sequences N varied on real dataset. We pick out queries from the query set of size 15 so that their lengths distribute uniformly. Note that no matter the value of N, we find out top-100 subsequences from the real time series. The results are shown in Fig. 6.

It can be seen that in Fig. 6(a), the accuracy of MQ, UCR-ED, and UCR-DTW increases when N increases while the accuracy of SpADe is consistently low. The reason is that SpADe extracts local patterns by using a fixed size of sliding window, and consequently, it fails to capture the query intuition. Moreover, it is noteworthy that when $N = 6$, the accuracy of MQ has already exceeded 0.9, which means that MQ is able to find out what users really want with only a small query set.

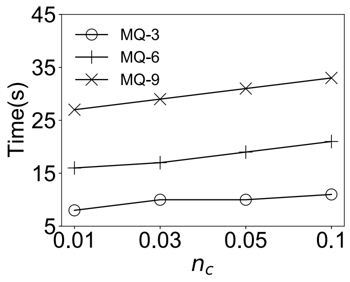
Figure 6(b) compares MQ and other approaches on efficiency. Obviously, MQ outperforms all other approaches, and its running time is insensitive to N since MQ summarizes all the queries and searches for results in the time series only once. Due to the specific pattern shape and the lack of abundant pruning strategies, UCR-ED is inferior to UCR-DTW in terms of efficiency in this dataset.

6.5. *Influence of Parameter n_c*

In this experiment, we investigate the influence of the parameter n_c, the size of the candidate set. On the real dataset, we varied n_c from 0.01 to 0.1, and the number of queries, N was set to 3, 6, and 9 respectively.



(a) Accuracy vs. n_c



(b) Efficiency vs. n_c

Fig. 7. Influence of parameter n_c.

Results are shown in Fig. 7. It can be seen that as n_c gets larger, the accuracy of MQ increases while the efficiency decreases. It is because once the query set is fixed, we can maintain more candidates by enlarging the candidate set, i.e. increasing the value of n_c at the cost of more running time. Also, we can find that MQ has already achieved satisfying results when $n_c = 0.05$, so we set the default value of n_c to 0.05 in previous experiments. Generally, as shown in Fig. 7(a), the accuracy of MQ increases as the number of queries N increases. The only exception is when $n_c = 0.01$. The reason is that the standard deviation of the query feature values experiences an increase as N increases, resulting in a larger search space. The small candidate set then fails to maintain enough candidates, and thus achieves poor results. Meanwhile, the changes in the search space cause the running time to increase proportionally, as shown in Fig. 7(b).

7. Conclusions

In this paper, we have proposed a novel subsequence matching approach, Multi-querying, to reflect the query intuition. Given multiple queries, we use a multi-dimensional probability distribution to represent them. Then, a breadth-first search algorithm is then applied to finding out the top-K most similar subsequences. Extensive experiments have demonstrated that Multi-querying outperforms the state-of-the-art algorithms in terms of accuracy and performance. To the best of our knowledge, this is the first study to introduce the concept of multiple queries to express the query intuition.

Funding

This work is supported by NSFC under grant 61962058, the Tianshan Talent of Xinjiang Uygur Autonomous Region – Young Top Talents in Science and Technology (2022TSYCCY0008), Integration of Industry and Education-Joint Laboratory of Data Engineering and Digital Mine (2019QX0035), Bayingolin Mongolian Autonomous Prefecture Science and Technology Research Program (202117), Natural Science Foundation of Xinjiang Uygur Autonomous Region (2019D01A30), and Scientific Research Program of the Higher Education Institution of Xinjiang (XJEDU2018Y056).

References

Abanda, A., Mori, U., Lozano, J.A. (2019). A review on distance based time series classification. *Data Mining and Knowledge Discovery*, 33(2), 378–412. https://doi.org/10.1145/3514221.3526183.

Boniol, P., Palpanas, T. (2020). Series2graph: Graph-based subsequence anomaly detection for time series. *Proceedings of the VLDB Endowment*, 13(12), 1821–1834. https://doi.org/10.14778/3407790.3407792.

Boniol, P., Linardi, M., Roncallo, F., Palpanas, T. (2020). Automated anomaly detection in large sequences. In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, Dallas, TX, USA, pp. 1834–1837. https://doi.org/10.1109/ICDE48307.2020.00182.

Boniol, P., Meftah, M., Remy, E., Palpanas, T. (2022). DCAM: dimension-wise class activation map for explaining multivariate data series classification. In: *Proceedings of the 2022 International Conference on Management of Data, SIGMOD '22*. Association for Computing Machinery, New York, NY, USA, pp. 1175–1189. https://doi.org/10.1145/3514221.3526183.

Chen, Y., Nascimento, M.A., Ooi, B.C., Tung, A.K.H. (2007). SpADe: on shape-based pattern detection in streaming time series. In: *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007*, The Marmara Hotel, Istanbul, Turkey, April 15–20, 2007, pp. 786–795. https://doi.org/10.1109/ICDE.2007.367924.

Dau, H.A., Bagnall, A., Kamgar, K., Yeh, C.-C.M., Zhu, Y., Gharghabi, S., Ratanamahatana, C.A., Keogh, E. (2019). The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6), 1293–1305. https://doi.org/10.1109/JAS.2019.1911747.

Eravci, B., Ferhatosmanoglu, H. (2013). Diversity based relevance feedback for time series search. *Proceedings of the VLDB Endowment*, 7(2), 109–120. https://doi.org/10.14778/2732228.2732230.

Hu, W., Letson, F., Barthelmie, R.J., Pryor, S.C. (2018). Wind gust characterization at wind turbine relevant heights in moderately complex terrain. *Journal of Applied Meteorology and Climatology*, 57(7), 1459–1476. https://doi.org/10.1175/JAMC-D-18-0040.1.

Iwana, B.K., Uchida, S. (2020). Time series classification using local distance-based features in multi-modal fusion networks. *Pattern Recognition*, 97, 107024. https://doi.org/10.1016/j.patcog.2019.107024. https://www.sciencedirect.com/science/article/pii/S0031320319303279.

Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S. (2001). Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3), 263–286. https://doi.org/10.1007/PL00011669.

Kondylakis, H., Dayan, N., Zoumpatianos, K., Palpanas, T. (2018). Coconut: a scalable bottom-up approach for building data series indexes. *Proceedings of the VLDB Endowment*, 11(6), 677–690. https://doi.org/10.14778/3184470.3184472.

Li, Y., Tang, B., U, L.H., Yiu, M.L., Gong, Z. (2017). Fast subsequence search on time series data. In: *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017*, Venice, Italy, March 21–24, 2017, pp. 514–517. https://doi.org/10.5441/002/edbt.2017.58. https://openproceedings.org/2017/conf/edbt/paper-369.pdf.

Linardi, M., Palpanas, T. (2018). Scalable, variable-length similarity search in data series: the ULISSE approach. *Proceedings of the VLDB Endowment*, 11(13), 2236–2248. https://doi.org/10.14778/3275366.3284968.

Muthumanickam, P.K., Vrotsou, K., Cooper, M., Johansson, J. (2016). Shape grammar extraction for efficient query-by-sketch pattern matching in long time series. In: *11th IEEE Conference on Visual Analytics Science* *and Technology, IEEE VAST 2016*, Baltimore, MD, USA, October 23–28, 2016. IEEE Computer Society, pp. 121–130. https://doi.org/10.1109/VAST.2016.7883518.

Papapetrou, P., Athitsos, V., Potamias, M., Kollios, G., Gunopulos, D. (2011). Embedding-based subsequence matching in time-series databases. *ACM Transactions on Database Systems*, 36(3), 17–11739.

Rakthanmanon, T., Campana, B.J.L., Mueen, A., Batista, G.E.A.P.A., Westover, M.B., Zhu, Q., Zakaria, J., Keogh, E.J. (2012). Searching and mining trillions of time series subsequences under dynamic time warping. In: *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'12*, Beijing, China, August 12–16, 2012. ACM, pp. 262–270. https://doi.org/10.1145/2339530.2339576.

Shieh, J., Keogh, E. (2008). ISAX: indexing and mining terabyte sized time series. In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '08*. Association for Computing Machinery, New York, NY, USA, pp. 623–631. https://doi.org/10.1145/1401890.1401966.

Wang, H., Li, C., Sun, H., Guo, Z., Bai, Y. (2018). Shapelet classification algorithm based on efficient subsequence matching. *Data Science Journal*, 17(6), 1–12. https://doi.org/10.5334/dsj-2018-006.

Wang, Q., Whitmarsh, S., Navarro, V., Palpanas, T. (2022). IEDeaL: a deep learning framework for detecting highly imbalanced interictal epileptiform discharges. *Proceedings of the VLDB Endowment*, 16(3), 480–490. https://doi.org/10.14778/3570690.3570698.

Wasay, A., Wei, X., Dayan, N., Idreos, S. (2017). Data canopy: accelerating exploratory statistical analysis. In: *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017*, Chicago, IL, USA, May 14–19, 2017, ACM, pp. 557–572. https://doi.org/10.1145/3035918.3064051.

Wu, J., Wang, P., Pan, N., Wang, C., Wang, W., Wang, J. (2019). KV-match: a subsequence matching approach supporting normalization and time warping. In: *35th IEEE International Conference on Data Engineering, ICDE 2019*, Macao, China, April 8–11, 2019. IEEE, pp. 866–877. https://doi.org/10.1109/ICDE.2019.00082.

Zoumpatianos, K., Idreos, S., Palpanas, T. (2016). ADS: the adaptive data series index. *VLDB Endowment*, 25(6), 843–866. https://doi.org/10.1007/s00778-016-0442-5.

W. Liu received the BS degree in computer science from Xinjiang Normal University, Xinjiang, in 2004, and the PhD degree in computer science from Dalian University of Technology, Dalian, China, in 2009. He is currently a professor in College of Control Engineering, Xinjiang Institute of Engineering. His research interests include database, stream data processing, and cloud computing.

M. Ma received the BS degree in computer science from Xinjiang Normal University, Xinjiang, in 2004, and the MS degree in computer science from Dalian University of Technology, Dalian, China, in 2012. He is currently a PhD student of computer science at Xinjiang University. His research interests include database and cloud computing.

P. Wang received the BS degree in mathematics from Nankai University, Tianjin, China, in 2001, and the PhD degree in computer science from Fudan University, Shanghai, China, in 2007. He is currently a professor in School of Computer Science, Fudan University. His research interests include database and stream data processing.