CIT. Journal of Computing and Information Technology, Vol. 28, No. 1, March 2020, 1–17
doi: 10.20532/cit.2020.1004574

Fast Online Similarity Search for Uncertain Time Series

Ruizhe Ma[1], Diwei Zheng[2] and Li Yan[2]

[1]Georgia State University, Atlanta, USA
[2]Nanjing University of Aeronautics and Astronautics, Nanjing, China

To achieve fast retrieval of online data, it is needed for the retrieval algorithm to increase throughput while reducing latency. Based on the traditional online processing algorithm for time series data, we propose a spatial index structure that can be updated and searched quickly in a real-time environment. At the same time, we introduce an adaptive segmentation method to divide the space corresponding to nodes. Unlike traditional retrieval algorithms, for uncertain time series, the distance threshold used for screening will dynamically change due to noise during the search process. Extensive experiments are conducted to compare the accuracy of the query results and the timeliness of the algorithm. The results show that the index structure proposed in this paper has better efficiency while maintaining a similar true positive ratio.

ACM CCS (2012) Classification: Mathematics of computing → Probability and statistics → Statistical paradigms → Time series analysis

Information systems → Information retrieval → Retrieval models and ranking → Similarity measures

Keywords: time series, uncertainty, online similarity search, index

1. Introduction

With the improvement of computing power and the development of data mining technology, time series is of growing importance in many new streaming applications, such as GIS detection [6], stock market [16] and medical monitor [28]. Thanks to the advances in large-scale storage and computing power, there have been a lot of efforts in exploiting large time series databases [16, 29, 33, 36]. Due to the factors such as physical equipment, calculation of distance metrics, and inherent noise of the sample, the streaming data always carry uncertainty. Generally, uncertainty leads to drawing erroneous conclusions. In order to keep results from great bias, the uncertainty information must be taken into consideration. For example, each timestamp is modeled as a random variable that associates a probability density function [13]. Most of current work, however, only concentrates on exact queries [5, 8, 23, 25, 26, 27, 33, 34] although uncertain queries rather than exact queries are greatly required. Especially, uncertainty is universal in emerging applications that deal with streaming series, object identification [4] and environmental applications [29]. Diverse uncertain objects have been extensively investigated. But, to the best of our knowledge, the first effort that proposes a framework for similarity matching of uncertain time series is presented in [13], where the uncertain time series is named cloaked time series and the fundamental query predicates are discussed. Furthermore, in [1], the notion of an uncertain time series is formalized, two novel kinds of probabilistic range queries are introduced, and a primitive approximate representation of uncertain time series is proposed.

Generally, there are three types of spatial queries over time series.

- Range query: Given a target point Q and a threshold ε, find all points X such that $Dist(Q, X) \leq \varepsilon$.
- Nearest neighbor query: Given a query point Q, find all points X such that $Dist(Q, X)$ is the minimum. Furthermore, the k-nearest neighbor query asks for the k closest points to a given point.

- Subsequence query: Given a query sequence Q, find all subsequences that satisfy $Dist(Q[i:j], X[m:n]) \leq \varepsilon$. For example, *DTW* (Dynamic Time Warping) calculates the similarities between sequences of different lengths based on dynamic programming.

Limited by the high dimension of the sequence, most current work just focuses on the uncertain offline data [7, 11, 17, 18, 19]. No efficient index has been proposed so far to support online query processing for uncertain time series. The computational complexity of distance metric is proportional to the sequence length. So, a sequence that is too long in the candidate set screening process will seriously affect the query speed.

To deal with time series with index technique, many efforts have tried to build a spatial index structure for querying the sequence data. In [21], a set of points is picked into a minimum bounding rectangle (*MBR*) and a spatial index *R-tree* is used to query the target *MBRs*. Generally, *R-tree* is mainly used to solve the problem of querying non-zero objects which are treated as a rectangle (namely *MBR*) for processing. To improve the efficiency of retrieval, in [31], the sequence is mapped into the frequency domain by using the Discrete Fourier Transform (*DFT*), and the first few large coefficients are kept in the *R-tree*. Unlike [31], the algorithm *FastMap* [25] tends to build the *KD-tree* by using time wrapping measurements. As shown in Figure 2, the *KD-tree* (*k*-dimensional) is a data structure that divides the *k*-dimensional data space. It is mainly used for the search of key data in multi-dimensional space (such as range search and nearest neighbor search). Collaborating with the time wrapping measurements, it can work well for the similarity matching over subsequence. At the same time, *R*-Tree* [24] further expands the type of problem as well as the deciding factors of *MBR* partitioning based on *R-tree*. Compared with *KD-tree*, the *MBR* in *R-Tree* is more flexible because the *MBRs* in *R-tree* can overlap with each other. Yet, *KD-tree* tidily splits space without a redundant or uncovered place on the same layer. The experiment confirms that, in terms of the efficiency of retrieval, *R*-Tree* outperforms *KD-tree* while in terms of the efficiency of update, *KD-tree* is superior to *R*-Tree*. To utilize the advantages of both, we propose an index *KDR-tree* by combining the structure of *KD-tree* with the retrieval algorithm of *R-tree*. Specifically, we put forward a search algorithm based on the *R-tree*'s query process, and adapt the split threshold of *KDR-tree* to limit the number of the split operations.

In this paper, we concentrate on the subsequence query and present a novel and efficient indexing technique over the online sequence. The work surrounds the subsequence query about uncertain time series, *i.e.*, finding the candidates satisfying $P(Dist(Q, X) \leq \varepsilon) \geq \tau$. In the matching of similar sequence, the traditional index is designed for the exactly matched sequence. Based on these indexes, the queries always generate imprecise distance measurement rather than the exact answer. When uncertain information is taken into consideration, the measurement for the probabilistic threshold can calculate the difference of the likelihood of each answer. The measurement *DUST* accommodates the uncertainty and generalizes the notion of measurement. We use *Euclidean* and *DUST* as distance metrics in *KDR-tree*.

Our contributions in this paper are summarized as follows.

1. We combine the characteristics of *R-tree* and *KD-tree* to build an index structure *KDR-tree* for online data.
2. We propose an adaptive segmentation algorithm to calculate the maximum number of accommodating points for the spatial split.
3. We have experimentally confirmed the superiority of *KDR-tree* in terms of performance and efficiency. *KDR-tree* can reduce the loss rate without the cost of the true positive rate.

The rest of this paper is organized as follows. The main symbols are listed in Table 1. In Section 2, we give a brief description of related work for uncertain time series. Section 3 presents the method to index online for the uncertain time series. We present how to quickly build an index of real-time data by combining the characteristics of *KD-tree* and *R-tree* in Section 4. The experiments are presented in Section 5. We finally conclude the paper in Section 6.

Table 1. Main symbols used in this paper.

Symbols	Description
X	The sequence consists of the primitive data
$\widehat{X}$	The sequence consists of the features extracted by *DFT*
$X[i{:}j]$	A sequence representing the value of the timestamp from i to j
$X[i]$	The i-th timestamp value
$\mu_i = E(X_i)$	The expected value of the i-th timestamp variable
T_i	The i-th timestamp real variable
$X_i = T_i + \delta_i$	The i-th timestamp output variable

2. Related Work

2.1. Discrete Fourier Transform

There are two main techniques for the decomposition of sequences. The first proposed technique is *DFT*, which is followed by the *Haar* decomposition [27]. Most of the research on feature extraction of the time series is based on both. In this paper, we use *DFT* to extract the feature of the expectation. The earliest work to build an index with an eigenvalue sequence is to construct the index structure based on the coefficients extracted by the *DFT* [31]. Its important contribution is to reduce the sequence dimension, which needs to be calculated by using the theorem that the value of the distance metric in the frequency domain space is equal to the distance of the original space. Because the *DFT* can find the most frequent eigenvalue in frequency space, only a few coefficients should be kept to approximate the original value. Here the dimensions of these eigenvalues are generally much smaller than the original dimension. This method can greatly reduce the amount of calculation in the metric. Meanwhile, the property of the triangle inequality of the distance measurement allows us to use optimization methods to search for candidates in the feature space as in the original space. In summary, we can combine the *DFT*, spatial index structure and uncertainty metrics to query for the candidates.

2.2. Sequence Matching Using Index

In [15], the *FRM* is used to extract the features of the subsequence, where the subsequence matching algorithm is proposed based on those features. The algorithm consists of the index building and subsequent matching. In the index building, the *FRM* divides time series into sliding windows of size w and stores feature points in a spatial access method, *R*-tree*. The structure of *R*-tree* is shown in Figure 1, which is mainly used to organize the *MBR*. In the subsequence matching, the *FRM* transforms the feature points into the f-dimensional points by using a feature extraction algorithm during the matching process. In [35], an improvement of the known *DFT*-based indexing technique is proposed, in which the first few large Fourier coefficients are used in the distance computation. The work in [32] measures the subsequence similarity by Time Warping (*TW*) based on those coefficients. After mapping the objects into the *k-d* points, a spatial access method, *KD-tree*, is used to organize them. Furthermore, a bounding technique is designed in [30] to prune the unnecessary computation as much as possible. To achieve Dynamic Time Warping (*DTW*), a novel algorithm *SPRING* is presented to select all matched subsequences. For the interval problem, the extensive index scheme is established based on the *R-tree* in [20]. For discrete data, the search algorithm *PROUD* is applied in [2] to a streaming uncertain time series. And the *Haar* wavelet decomposition is used to construct an error-tree to retrieve the distance measurement efficiently.

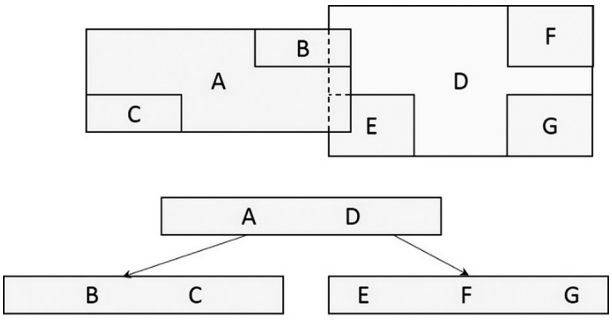


Figure 1. R-tree.*

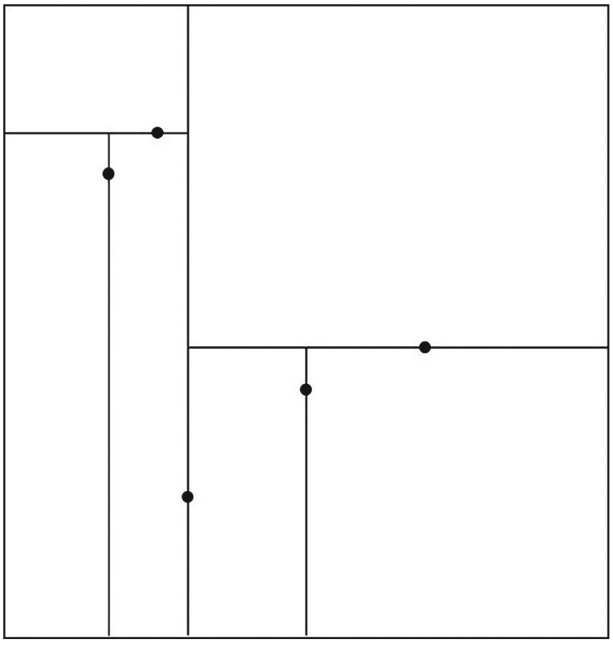


Figure 2. KD-tree.

2.3. Uncertain Similarity Measurement

The *PROUD* presented in [2] is a probabilistic approach to processing similarity queries over uncertain streams. Although this approach is flexible to control the trade-off between false positives and false negatives, it cannot be quantified. The approach is only suitable for the specified uncertain time series, where each timestamp has the same variance. Moreover, the filter function is limited to the Euclidean distance. When the measurement is changed into another distance, such as the *TW*, the filter function must change as well. A theoretical framework is applied in [3] to generalize the notion of similarity between uncertain time series. Furthermore, a novel distance measure *DUST* is proposed, which takes uncertainty into consideration. While the *PROUD* can only measure the sequences with the same noise variance and the metric is still not quantifiable, the *DUST* goes one step further on the basis of *PROUD*. The algorithm can process diverse uncertainties. Variance of the noise of each timestamp can be a different type. However, it is too time-expensive to be feasible for streaming data applications. As for the *DTW*, it aims to find the optimal alignment with a minimum distance by typically using a dynamic programming technique. To adapt the *DTW* to the real-time and streaming data, the *SPRING* algorithm is proposed in [40] to dramatically improve the naive method. Unnecessary computation is pruned in [30] by using a bounding technique, which can accelerate the process by at least three times, compared to the *SPRING*.

3. Proposed Method

To construct the index over the streaming uncertain series, we use the subsequence distance measurement. Here we present the concrete question definition as follows:

3.1. Problem Definition

Suppose that the user specifies a target time series Q of length $Len(Q, T)$ at time T. Let the probability threshold and the tolerance of the difference of the distance be ϵ and ε, respectively. Here, T is the time when the user wants to get access to the history series.

Assume that we have a collection of $Num(T)$ sequences of real time series $S_1, S_2, ..., S_{Num(T)}$, each one of a potentially different length. We aim to efficiently get all qualified candidates $S_i (1 \leq i \leq Num(T))$, along with the specified offset k, such that the subsequence $S_i [k{:}k + Len(Q, T) - 1]$ matches the query sequence: $P(Dist(S_i [k{:}k + Len(Q, T) - 1], Q) \leq \varepsilon) \geq \tau$.

3.2. Proposed Approach

For uncertain time series data, different models have been proposed. This paper mainly uses the models based on probability statistics. This means that the data at each moment is composed of the distribution of real data T and the

distribution of noise δ. Hence each timestamp is a random variable and corresponds to a mean and to a variance. As shown in Figure 3, the distribution of a timestamp will be (μ_i, σ_i) where μ_i is a mean and σ_i is a variance.

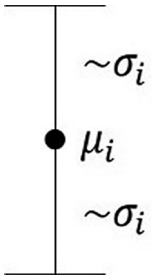


Figure 3. The distribution of one timestamp.

We define X as a reference series with uncertainty and Y as one of the subsequences with the uncertainty stored in the database. Both series consist of the random variables in each time series. Specifically, $X[i] = T_x[i] + \delta_x[i]$ and $[i] = T_y[1] + \delta_y[i]$. After pretreatment with the *DFT*, we get $\widehat{X}$ and $\widehat{Y}$ series that are composed of features extracted from the X and Y and satisfy the following equation.

$$\sum_{i=1}^{M}\left(\widehat{X}[i]-\widehat{Y}[i]\right)^2=\sum_{i=1}^{N}\left(X[i]-Y[i]\right)^2 \quad (1)$$

Here M is the dimension of the feature space and N is the length of the original sequence. The *FRM* gets the sub-trails after mapping the series with an allowable length into a feature space. Based on those features, the spatial index is constructed in real time. Since the sub-trails are included in the *MBR*, there will be no false dismissal. However, the cost function with respects to the *MBR* margin, which is one of the factors to decide how to split the *MBR*, is always time-expensive. Thus, we propose a method that can avoid this condition. We put forward the *KDR-tree* to index the *DFT* coefficients based on the local linear assumption. We split the feature space along the dimension without crossing any points to construct the *KDR-tree*. While the *R*-tree* is used in [15] to insert the offline data into the new *MBR* in advance, the *KDR-tree* is constructed dynamically for the online data. Meanwhile, the max number K of points included in the *MBR* is determined by a number of factors and is adaptive to control the volume of the feature space. We define the K determination equation as follows.

$$K = T(area, variance, density) \quad (2)$$
$$= \frac{1.5\,K_{last}}{1+exp\left(-sign(area-area_{last})\frac{variance\cdot density}{area}\right)}$$

Here K_{last} represents the parent's or the child's number of current K and $area_{last}$ represents the parent's or the child's number of *area*. If the split operation takes place, K_{last} is the parent's K and the $area_{last}$ is the parent's K. On the other hand, if the index removes the last point of a subspace and combination happens, K_{last} is the child's K and the $area_{last}$ is the child's *area*. As for the root node, we need to manually set an initial value for K. The density is obtained by dividing the number of points surrounded in space by the volume of space. As we can see, the new K value is obtained by multiplying the K value of its parent/child node by a constant. The constant is calculated based on the logistic function. Its changing image is shown in Figure 4 It is shown that the K value of the new space is increased or decreased by half at most.

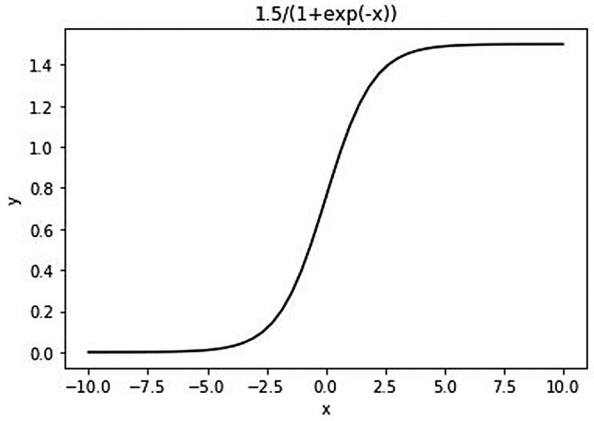


Figure 4. Adaptive K function.

Given the incoming point sets $US(T) = (S_{1_2,\ Len(S_1)}, \ldots, S_{M_i,\ Len(S_{M_i})})$ at time T, we should update the *KDR-tree* over $US(T)$. We add a new boundary to split the *MBR* when the number of points in the *MBR* is bigger than K. The split operation only affects the subspace that intersects the *MBR* formed by the newly arrived point set. Meanwhile, we need to update the K value of the new sub *MBRs* in real time. The *KDR-tree* divides the existing *MBR* instead of creating an

MBR to store the new points. Compared with the *R*-tree*, it saves the split and inserts time because it doesn't enumerate all sub *MBRs* to determine the best inserted *MBR* and calculate the area of sub *MBRs*. Thus, the index can quickly respond to the coming data. Because of this fast response to real-time data, the index structure has high throughput. Figure 5 shows the content of the *MBR* we use. The leaf *MBRs* of the *KDR-tree* store the set of points in the subspace while the non-leaf *MBR* only saves the information about the split boundary of the parent space. The point set of the subspace is highly repetitive, hence there is no need to store the points in the non-leaf *MBR*. Moreover, the subspace size can be obtained along the path from the top down.

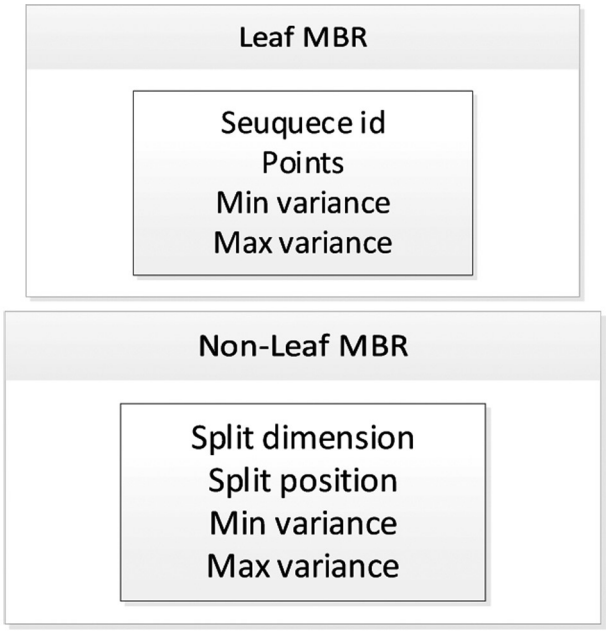


Figure 5. Data structure of *MBR*.

The *KDR-tree* uses the split algorithm, which is similar to the *KD-tree*, but the point is only included in the leaf nodes and no point passes through the split boundary. The biggest difference is that this is applied to uncertain time series. As for the maximum number of points, it is allowed to be adaptively changed according to the density and area as well as variance.

We give a simple example of *KDR-tree* in 2-dimensional space. Figure 3 shows a possible case of spatial division. Each of the subspaces has the maximum number of points equal to 1.

We define *K* as the formula (3) because the density distribution of the index is always uneven which is not beneficial for the search efficiency. For example, a small area containing as many points as a large area is easier to cause unnecessary depth exploration during the search. Hence, we define the density value as the ratio of a point set to the area and we consider it to be one of the standards for splitting the space. The higher the density, the easier it is to be divided. Meanwhile, given a search range, it is more possible to eliminate unnecessary space lookups for space with a large area. Otherwise, if the area itself is small, the same search radius is likely to cover the entire space and this makes the enumeration unavoidable. Finally, due to the larger variance, the denser the point set is within the point cluster, the more obvious is the separation between the points. It means that the space with a large variance should be easier to segment. Hence, we make *K* inversely proportional to the variance along the split dimension.

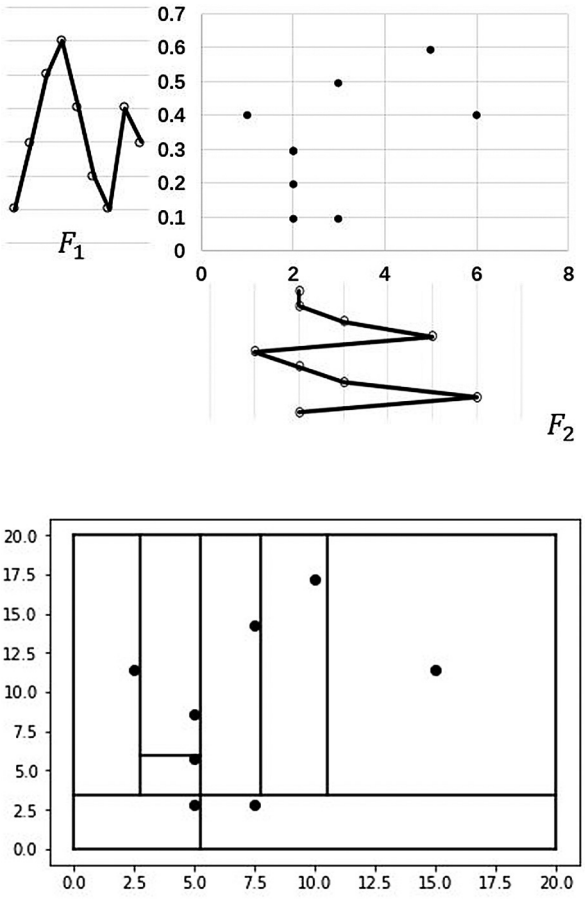


Figure 6. Example of mapping the series into feature space.

In summary, the non-leaf node is used to find candidates satisfying $P(Dist(S_i[k{:}k + Len(Q, T) - 1], Q) \leq \varepsilon) \geq \tau$ and the leaf node returns to the target points. The search spends a lot of time on these nodes to check the distance measurement rather than update the index structure in real time.

4. Index Construction

In the previous section, we discussed the sketched approach to dealing with the updating points. We used an inserting operation similar to the *KD-trees* operation, to avoid excessive *MBR* updates. In this section, we present how to construct the spatial index. First, we select the *DFT* as our feature extraction method used by the subsequence. Second, we discuss the search and the algorithm when the new points arrive.

4.1. Feature Extraction

We choose the *DFT* for three reasons.

1. It is commonly used for dimension reduction.
2. It provides a good and intuitive example to make the presentation clearer.
3. It keeps the distance measurement the same as the real distance, except for the white noise.

Furthermore, the feature extraction is important for us to reduce the length of the series resulting in higher efficiency when we treat the time as one dimension. Meanwhile, it is useful when the distance measurement over an uncertain time series is complicated. After the updating set arrives, we check if the space needs to be divided according to *K*.

4.2. Search

The search operation is used to select the qualified candidates when the user gives a query series at a specific time. At this point, we must choose one distance measure with uncertainty taken into consideration. Although the *PROUD* approach formalizes a selection standard by using a normalized function, it cannot be used directly to compare the distance in the form of numeric. Hence, in this paper, we use a variant numeric approach based on the *PROUD* as follows.

- The user query Q is mapped into the sub-trails in the same manner. The *KDR-tree* works for the specific area, which satisfies $P(Dist(S_i[k{:}k + Len(Q, T) - 1], Q) \leq \varepsilon) \geq \tau$.
- We use the variant distance measurement based on the *PROUD* to collect the candidates from the top down.
- We combine the results of the collections of the leaf points according to the identified *id* in the *MBR* over each uncertain time series.

In this paper, we focus on the uncertain time series. This means that our search process will take more time due to the addition of uncertain information. Correspondingly, the distance measurement is based on uncertain objects rather than on the crisp ones. For the continuous model, the value consists of real value and noise. Both of them are regarded as variables linked with an unknown probability density function. Although the distribution over a single random variable is unknown, the Central Limit Theorem indicates that the group of variables can be simulated by the *Gauss Distribution*.

We use the *KDR-tree* to perform the search actions on uncertain time series and it is different from the process of screening for crisp time series. Although both are based on Euclidean distance, the screening radius in the *KDR-tree* changes dynamically during the algorithm execution.

According to [41], the varying threshold (*i.e.*, screening radius) can be represented by the average of the random variable (*i.e.*, $E(x_i)$). Since the measurement is based on the probability theory, it must utilize the characteristics of the expected value as the computational element of the Euclidean distance. Here, we directly present the result in [41]. The filter function can be reorganized as the following equation:

$$\sqrt{\sum_{i=1}^{N}\left(\mu_i - \mu_i^q\right)^2} \leq \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad (3)$$

We define $a = 1$, $b = r\text{-}limit^2 \sqrt{4\left(\sigma^2 + \left(\sigma^q\right)^2\right)}$,

$c = -\varepsilon^2 + N(\sigma^2 + (\sigma^q)^2)$. Here, μ_i is the expectation of the i-th timestamp of the sequence whose variance is σ in the database; μ_i^q is the expectation of the i-th timestamp of the query sequence whose variance is σ^q; N is the dimension of the feature space. The symbol *r-limit* is the threshold that satisfies the target probability value under a normal distribution, which is defined as follows:

$$r\text{-}limit = \sqrt{2}\, erf^{-1}(2\tau - 1) \tag{4}$$

Here, $erf^{-1}(x)$ represents the inverse of the error function; τ is the user-defined probability threshold; r is the user-defined distance threshold. According to the algorithm proposed in [41], the threshold dynamically changes with respect to the square of variance.

$$\begin{aligned} &VarThd(\sigma^2) = \\ &= \sqrt{\left(r\text{-}limit^2 - len\right)\left(\sigma^2 + \left(\sigma^r\right)^2\right) + r^2} - \\ &\qquad - r\text{-}limit\sqrt{\sigma^2 + \left(\sigma^r\right)^2} \end{aligned} \tag{5}$$

The symbol *len* in this paper can be considered equivalent to the dimension of the feature space since (1) the request sequence with a large length can be split into multiple short sequences and mapped to low dimensional space and (2) under the real-time environment, the proportion of short-sequence data is relatively large due to factors such as speed, throughput, and transmission limitations.

Furthermore, this threshold has monotonic nature respect to σ when *r-limit* is positive, otherwise it is uncertain. Therefore, for the sake of simplicity, we suppose *r-limit* > 0 or $\sigma^2 > \dfrac{r\text{-}limit^2 r^2}{r\text{-}limit^2 - r\text{-}limit \cdot len^2}$ to make sure the threshold is monotonic decrease reference to the σ^2. As we can see, the radius of the sieve inspection is the dynamic distance with respect to the variance. This property certainly increases the time of calculating the distance measurement. However, after the *DFT*-based feature extraction and the *KDR-tree* processes, it effectively reduces unnecessary point set checks and time-consuming updates. In [41], it is proposed to apply a one-dimensional variance interval as an *MBR* to construct an *R-tree*. Similarly, we store the set of points in the sub-feature space to the *MBR*, store the interval value of the variance into the feature space, and use the two filter mechanisms in [41] in the search. Since the arrival of data in the process of building a *KDR-tree* is random, the structure of the final tree will be uncertain.

The metrics for uncertain data are expanded in [3]. For more complex types of uncertainty, they propose the *DUST* metric, which is primarily measured based on the definitions below.

$$DUST(X,Y) = \sqrt{\sum_{i=1}^{N} dust\left(X[i], Y[i]\right)} \tag{6}$$

Probability information is mainly integrated into the *dust*.

$$dust(x,y) = \sqrt{-\log\left(\phi\left(|x-y|\right)\right) + \log\left(\phi(0)\right)} \tag{7}$$

In the paper, we use the *DUST* distance metric based on the error function, which is normally distributed. In the case that the noise data is *Gaussian Distribution*, the distance of *DUST* is only proportional to the absolute value of the difference. The *DUST* is essentially based on *Euclidean* distance and the *DFT* does not change the *Euclidean* distance between the original points. Hence, we can directly calculate the metric in the new feature space.

$$DUST(X,Y) = \frac{\sqrt{\sum_{i=1}^{N}\left(X[i] - Y[i]\right)^2}}{2\sigma(2+\sigma^2)} \tag{8}$$

After obtaining the candidate set, we use a post-processing method to get the last submitted candidate set. The **Search** algorithm (Algorithm 1) implements the steps described above.

4.3. Update

The update processing includes the addition of data sets and the partitioning of feature space. For the addition of data sets, the time complexity is constant since it is highly efficient to update the feature subspace. For the division of

the feature space, it is a bit cumbersome, mainly in checking all feature space intersecting the MBR formed by the set of points to be added. In summary, the update of the data includes the following steps.

- When the data to be updated arrives, we save it directly in the database.
- We calculate the *MBR* of these point sets and then find the feature space that intersects with the space in the *KDR-tree*.
- Based on the found *MBRs*, we determine if to divide the corresponding space by comparing the number of points and the value of K in each subspace after adding the corresponding point set.

For determining the split dimension, we use the variance as the criterion for screening the optimal dimension because the larger the variance, the more obvious the point set distribution. Not only do we need to consider variance, but also the size of the subspace. Intuitively, we know that, for the same search radius, if a circle corresponding to the radius covers a denser set of points and fewer nodes, there will be more points that may need to be enumerated. It means that a set of points with low density can help reduce unnecessary further node exploration. Conversely, a subspace with dense points is likely to cause unnecessary traversal due to insufficient filtering. Hence, we need to use the size of the space and the spatial density as a correlation factor. In short, we set three quantitative values, namely *variance*, *density* and *area*, as criteria for the division.

Most of the time is usually spent in searching and enumerating the space to be divided. Hence, compared to the *R-tree*, there is more time-cost for the *KDR* to determine if it needs to be split than to perform the splitting operation.

Above all, we have the following algorithm to update the *KDR-tree*. *TryToSplit* algorithm (Algorithm 2) is applied to determine whether the node needs spliting or not. And *ADD* (Algorithm 3) is an interface used to insert the new points.

Algorithm 1. Search.

Input: root R of *KDR-tree*, Q query feature, σ query variance, ε distance threshold, τ probability threshold
Output: *candidates*

1. **initialize:**
2. $r\text{-}limit \leftarrow \sqrt{2}erf^{-1}(2\tau - 1)$
3.
4. minThd = *getThreshold*(*r-limit*, *R.Dim*, $R.\sigma_{max}$, σ, ε)
5. maxThd = *getThreshold*(*r-limit*, *R.Dim*, $R.\sigma_{min}$, σ, ε)
6.
7. **if** max*Dist*(R.MBR, Q) ≤ minThd **then**
8. *GetCoveredLeaves* $\overset{\text{push}}{\Rightarrow}$ *candidates*
9. **end if**
10. **if** *minDist*(*R.MBR*, *Q*) > *maxThd* **then**
11. *return*
12. **end if**
13. **if** *R* is leaf node **then**
14. **for** i = 1, ..., *R.pointsNum* **do**
15. *thd* = *getThreshold*(*r-limit*, $R.\sigma[i]$, σ, ε)
16. **if** *Dist*(*R.points*[i], *Q*) ≤ *thd* **then**
17. *R.points*[i] $\overset{\text{push}}{\Rightarrow}$ *candidates*
18. **end if**
19. **end for**
20. *return*
21. **end if**
22. **if** (*R.points*[*R.splitDim*] − *R.splitVal*)^2 ≤ *maxThd* **then**
23. *Search*(*R.left*, *Q*, σ, ε, τ)
24. *Search*(*R.right*, *Q*, σ, ε, τ)
25. **end if**

Algorithm 2. TryToSplit.

Input: root R of *KDR-tree*
1. **initialize:** *curNode* = *R*
2. **if** *Count*(*R.points*) > *R.K* **then**
3. **for** *all dimensions* **do**
4. calculate the variance of two subspace
5. **if** the sum of two variances is the largest one **then**
6. **else** we choose this dimension to split
7. **end if**
8. **end for**
9. calculate the *area*, *density* of two subspaces
10. choose the median value as the split value along the choosen dimension
11. Update *R.left.K* and *R.right.K* according to *T*(*area*, *variance*, *density*)
12. **end if**

Algorithm 3. Add.

Input: root *R* of *KDR-tree*, *point* to be added
1. **initialize:** *curNode* = *R*
2. **while** *true* **do**
3. **if** *curNode* is leaf node **then**
4. insert the node into *curNode*
5. *TryToSplit*(*curNode*)
6. *return*
7. **end if**
8. **if** *point*[*curNode.splitDim*] < *curNode.splitVal* **then**
9. *curNode* = *curNode.left*
10. **else**
11. *curNode* = *curNode.right*
12. **end if**
13. **end while**

5. Experiments

In this section, we conduct experiments with random and real data, respectively. To determine if the adaptive *K* value improves efficiency of the index, we need to compare the search algorithm with the *KR-tree*. At the same time, to compare the influence of different distance metrics on the retrieval results, we compare the search algorithm based on the *PROUD* and the *DUST* distance metrics. In general, the *DUST* has a higher accuracy and a lower false detection rate since it makes full use of the uncertain sequence. By the experiments with different search algorithms we compare the efficiency of these algorithms. To facilitate comparison, we give two quantitative indicators in Table 2.

Table 2. Quantitative indicators.

		Candidates	
		True	False
Real Sequence	True	*TP*	*FN*
	False	*FP*	*TN*

$$Error\ Ratio = \frac{TP}{TP + FP}$$

$$Miss\ Ratio = \frac{TP}{TP + FN}$$

Our experiments were implemented in Python 3.7 and run on the PC with 2.4 GHz CPU and 4GB RAM.

5.1. Synthetic data

Time series data contain the noise distribution and the real distribution. We hereby set these two types of distributions to generate synthetic data. In the experiments, it is assumed that the noise distribution of data is a standard normal distribution and the true distribution is a uniform distribution. So, the expected value of each data is the expected value of the real data distribution. We further assume that the expected value is a uniform distribution with respect to time and the variance is a uniform distribution with respect to the sequence. This means that the different timestamps in the same sequence have the same variance but different mean values while different sequences have different variances. Once a new sequence data is generated, we first insert it into the index structure and then query the candidate set. Since the experiments use different index structures, the dimensions of each index structure are slightly different. Specifically, after feature extraction using the *DFT*, two feature values of the subsequences are generated.

The *KDR-tree* and the *KD-tree* require the feature's dimension to handle point sets, and we hence map subsequences to 4-dimensional points. On the contrary, the *R-tree* can work directly in the 2-dimensional space. To deal with real-time data, we focus on the construction of index and query. The *KD-tree* takes each point as a node and each node includes a dividing line. This means that, as the set of points increases, the space becomes more fragmented. At the same time, the depth of the tree will increase accordingly. Although the *R-tree* is more flexible than the *KD-tree*, the space of the *R-tree* can be more time-consuming because random *MBR* can overlap with each other. The *R-tree* is an index for the data of non-zero size and the update operation on the *MBR* is too time consuming to meet real-time performance. To observe the efficiency of different index updates and retrievals, we experiment with an update and an index operations. We add the synthetic data to different index structures and then use

the corresponding search algorithm to find the candidate set.

All experiments were repeated 20 times. Each experiment (randomly) generated 10 batches of random data under the corresponding test parameters to simulate real-time data. We averaged all test results as the final elapsed time. Here, an experiment was carried out under different numbers of processed objects to test the efficiency of different indexes. The final result is presented in Figure 7.

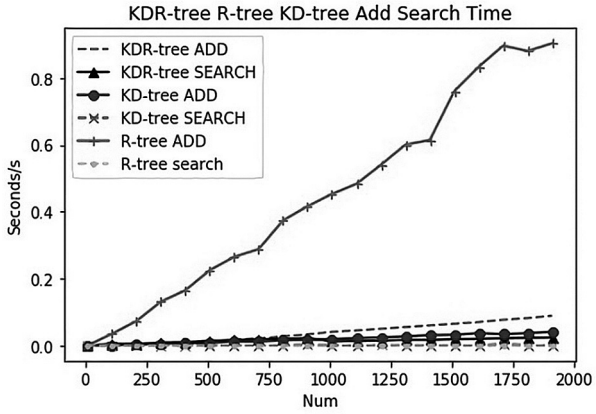


Figure 7. Efficiency comparison over the num.

It is shown in Figure 7 that, as expected, three index structures have their own advantages. Due to the flexibility of the *MBR*, the *R-tree* needs to update the *MBR* from the bottom up when the inserting operation takes place. Hence, it is time expensive to insert the coming data. On the contrary, due to the flexibility of the *MBR*, we can quickly locate the *MBR*, where the target is located when the *R-tree* is used for retrieval. Therefore, the *R-tree* is the most efficient in terms of retrieval, but it is the least efficient when updating. The *KDR-tree* query efficiency is between the *KD-tree* update and query efficiency. The structure of *KDR-tree* is similar to the *KD-tree*, and the space is divided into subspaces that do not overlap with each other. Therefore, there is no need to update a large number of *MBRs* when inserting data. Unlike the *KD-tree*, the *KDR-tree* needs to divide the space according to a series of rules. The division decision is based on the density, the area and the variance along the split dimension. Therefore, the division of *KDR-tree* increases the time of update, but a reasonable division can make the retrieval efficiency almost unchanged or even improveed. Due to the split decision, the depth of the *KDR-tree* does not increase linearly with the data points. Most importantly, the *KDR-tree* can handle uncertainty time series.

Concerning the effect of noise variance on query performance, performance of the index varies slightly under different noises. The experimental data is based on the random data generated under fixed τ and ε. Each corresponding noise variance has 10 sets of random data for testing. Similarly, we took the average of 10 sets of results as the final result. The experimental results are shown in Figure 8.

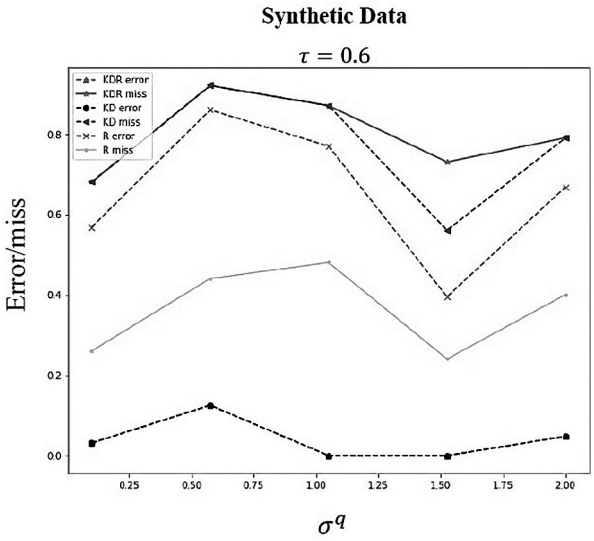


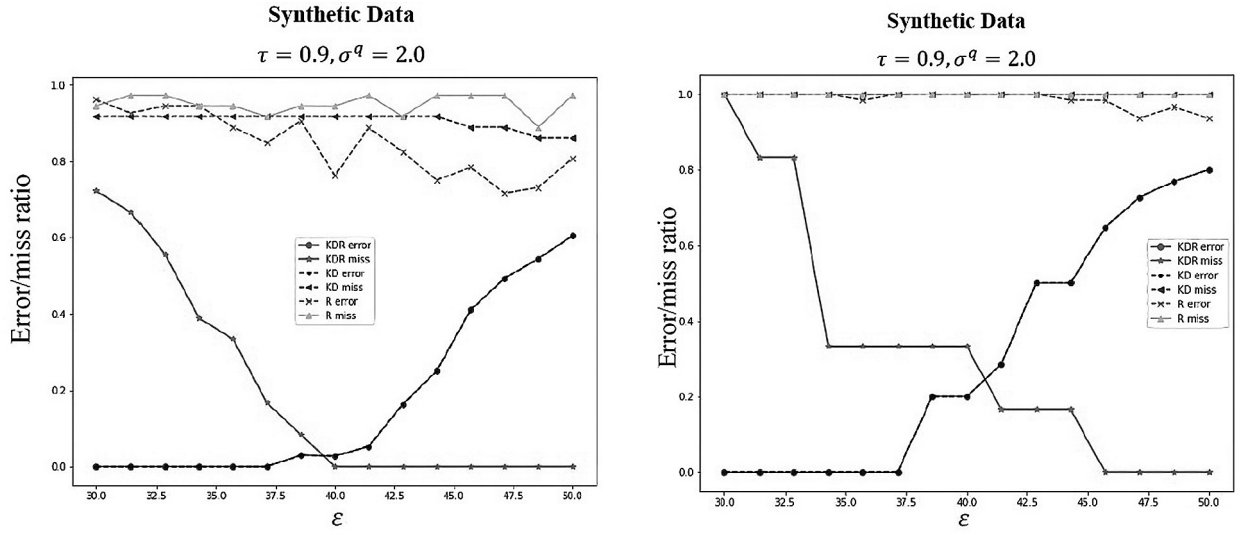
Figure 8. Error and miss ratio over σ^q.

In the result, the *KD-tree* and *KDR-tree* curves coincide because the *KDR-tree* retrieval step in the experiment is very similar to THE *KD-tree*. The difference is that the *KDR-tree* needs to traverse the leaf nodes so that the error rate may be less than or equal to the error rate of *KD-tree*. Moreover, the fast screening algorithm proposed in [16] can be used for the *KDR-tree*. It is shown that the error ratio and the missing ratio of the *R-tree* are biased towards 0.5. In the *R-tree*, the eigenvalues are organized into *MBR* forms. Unlike the *KD-tree* and the *KDR-tree*, the *R-tree* uses these *MBRs* instead of the points for retrieval. Therefore, the *R-tree* measurement result is smaller, causing the error value to be larger than the *KD-tree* and *KDR-tree.* On the contrary, the loss rate is relatively small.

Next, we need to test the impact of the distance threshold ε on the performance. Random data is generated based on the same behavior as the above experiment. Similarly, τ and σ are fixed during the experiment. Each ε corresponds with the average of the 10 experiments. Each experiment was carried out in the following context.

1. **Data Collection.** Suppose the database collects data per 30 seconds. Experiments randomize the number of each sequence in 30 seconds. Finally, we return the true value of the data to be updated, the noise value, the expected value and the variance of the observed sequence, through the simulated sensor.
2. **DFT Preprocess.** After collecting the data, we use the *DFT* to extract features. The experiments in this paper use 2-dimensional features. Moreover, the feature value is complexity. Hence the *KD-tree* and *KDR-tree* are constructed in 4-dimension space. The *R-tree* is built in 2-dimension space.
3. **Index Update.** After the point set of the feature space is obtained, the index structure is updated by using the insert operation of the corresponding index. The inserting may cause the node to be split.
4. **Search.** We randomly generate the subsequence to be queried (the subsequence is shorter to ensure that the distance metric of the 2-dimensional eigenvalue is close enough to the true value). After extracting the features of the subsequence, we use the updated index structure to retrieve the set of points that match the objective function $P(Dist(S_i[k{:}k + Len(Q, T) - 1], Q) \leq \varepsilon) \geq \tau$.
5. **Calculation of Statistics.** Based on the real data sequence, we calculate the number of subsequences that satisfy the objective function. Based on the candidate set obtained from the index structure, we filter the correct candidate set and the wrong candidate set. Eventually, we calculate the error ratio and the loss ratio.

Following the experimental step, we set up different ε for the experiments. Experimental results containing test results under the *DUST* and *PROUD* distance metrics are shown in Figure 9. Note that the error rates of *KDR-tree* and *KD-tree* overlap here. With the increasing threshold, the error rates of *KD-tree* and *KDR-tree* increase while the loss rate of all index structures decreases rapidly. On the contrary, the error rate of *R-tree* drops slightly. It is also shown that the *R-tree* gives a lot of candidate sets, but there are a lot of erroneous data in the candidate set. Both the *KDR-tree* and *KD-tree* give a small number of candidate sets, where the correct candidate takes a large proportion. These differences



(a) PROUD *(b)* DUST

Figure 9. Error and miss ratio over ε.

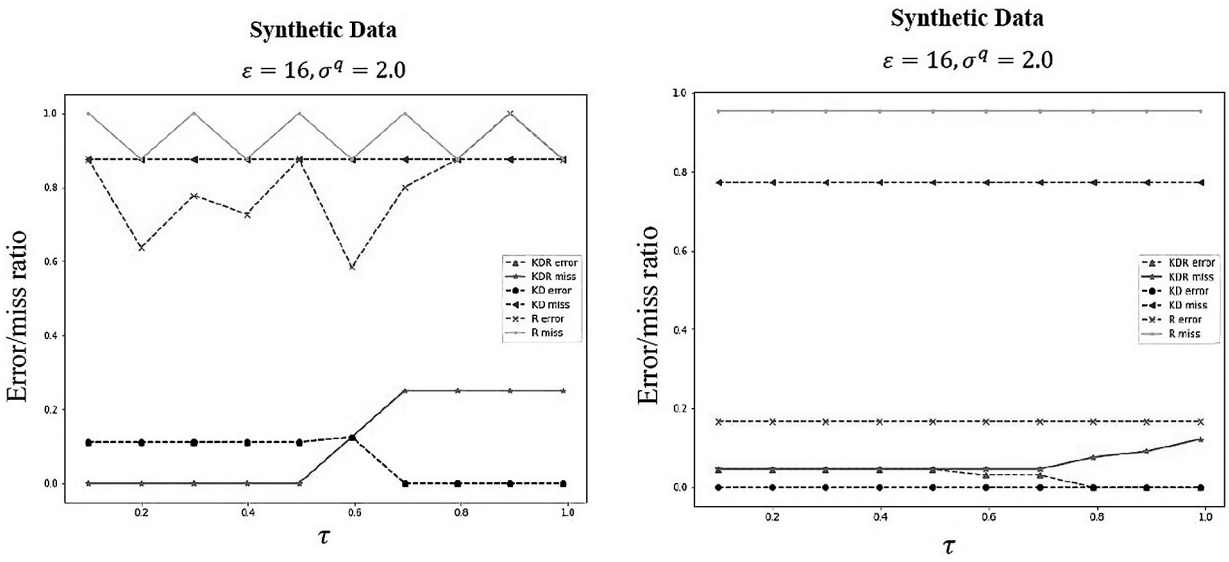
are mainly due to the fact that the point-based index construction and the *MBR*-based index construction query methods are very different. The *MBR*-based query underestimates the actual value of the distance metric, but the query based on the point set is closer to the true value. Hence, the result shows that the error rate of the *R-tree* is higher. Meanwhile, the *KDR-tree* has a lower loss rate than the *KD-tree*. This means that the overall performance of the *KDR-tree* is better than that of the *KD-tree*.

To fix the parameter ε, we compare the performance of different indexes on τ. The experimental results are shown in Figure 10. As [17] pointed out, the *PROUD*-based screening works only in a small interval of ε. This means that the choice of ε becomes very important. This is why we chose ε as 16. As τ continues to increase, the error rate remains the same or decreases. On the other hand, the miss ratio remains the same or increases. It is shown that τ has far less impact on the results than ε. This is because τ's influence in the filter function is small, and the distance that is ultimately used for filtering is linear about ε. As shown in Figure 10, the index structure changes very stably. As expected, the error rate increases as τ increases. The reason why the *R-tree* does not meet our expectations is the fact that the *R-tree* underestimates the metrics.

5.2. Real Data

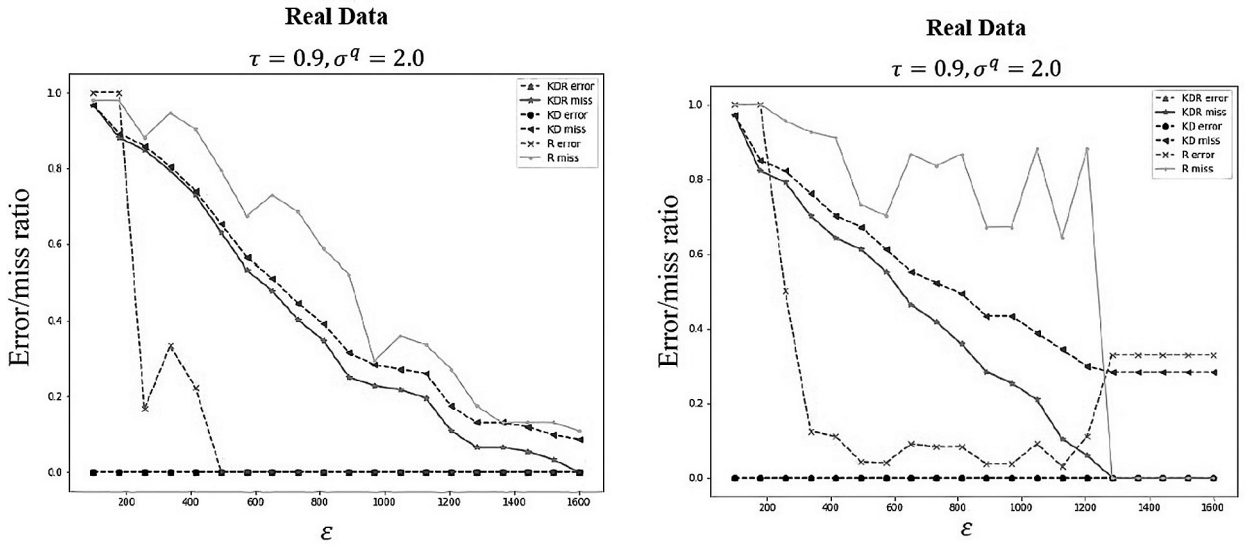
Archive files contain daily average temperatures of 157 U.S. and 167 international cities. Source data for these files are from the Global Summary of the Day (*GSOD*) database archived by the National Climatic Data Center (*NCDC*). The daily average temperatures posted on this site are computed from 24 hourly temperature readings in the Global Summary of the Day (*GSOD*) data. The data fields in each file are month, day, year, daily average temperature (*F*) and data containing "-99" no-data flag is not available. Since the pre-processed point set after the *DFT* is of the same dimension, we do not need to care about the length of the sequence. There are more than 300 cities in the original file and about 5,000 data points in each city that are not empty.

Unlike the synthetic data, we read a random number of non-null data from a file at the same time interval. In the experiments, we read the data for 300 cities every 30 ms within 2 minutes. If reading a file is finished early in 2 minutes, we do nothing. Since the fluctuations of dimensions and values of the real data are larger than those of random data, the distance threshold is increased correspondingly. We still examine the impact of the distance threshold ε first. Then we test the impact of τ. The results are shown in Figure 11 and Figure 12, respectively.



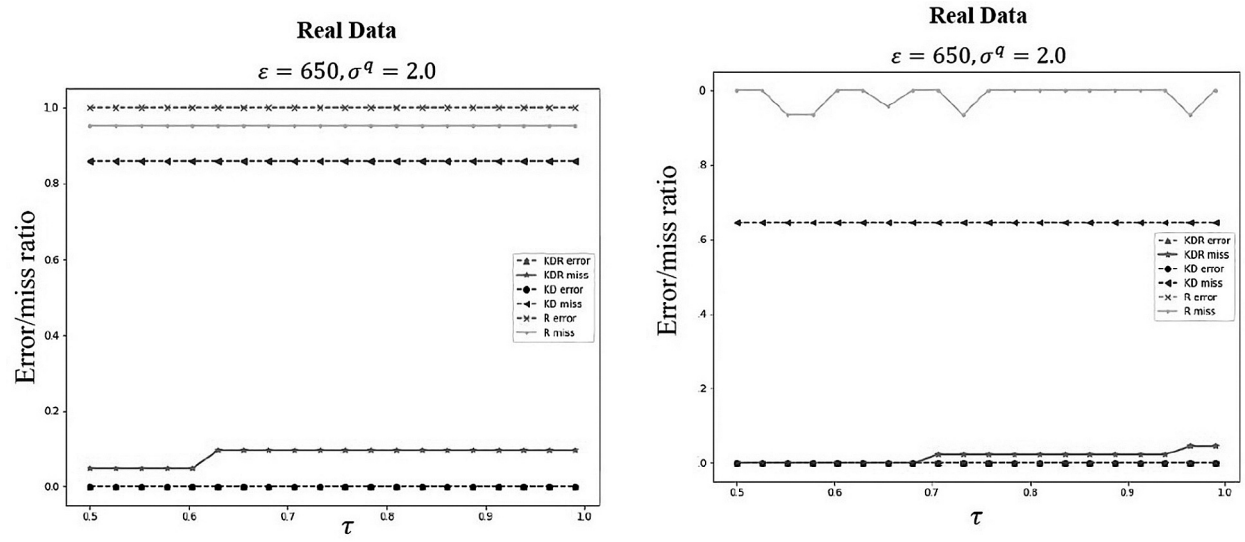
(a) PROUD *(b)* DUST

Figure 10. Error and miss ratio over τ.



(a) PROUD *(b)* DUST

Figure 11. Error and miss ratio over ε.



(a) PROUD *(b)* DUST

Figure 12. Error and miss ratio over τ.

The result is similar to the case of the synthetic data. The loss rates of these three index structures are all rapidly reduced. The reason why the error rate of *R-tree* rapidly decreased is that the *R-tree* does not introduce the wrong candidate set at the same time when the candidate sets are introduced. It is shown in Figure 9 that the loss rate of the *KDR-tree* is still superior. In the case of using the *DUST* distance metric, the loss rate of the *R-tree* becomes unstable.

Finally, it is shown in Figure 12 that τ has a smaller impact on the results than ε does. The reason is that, as the length of the sequence increases, the proportion of ε in the filter function takes up more and more. Therefore, performance of the index structure on τ seems to be stable. *R-tree* has a high error rate and a high loss rate due to excessive data acquisition. Once again, it is confirmed that the overall performance of the *KDR-tree* is better.

6. Conclusion

In this paper, we use the *DFT* to map the sequence into the feature space and then construct the index structure in the control space for comparison. Since the *DFT* guarantees invariance of the metric, the index can obtain the metric directly, based on the eigenvalues obtained by the query. The experimental results show that the *R-tree* is not suitable for dealing with online data. Although the *R-tree* retrieval is faster, the update operations required by the *R-tree* are too complex. On the contrary, although the update of the *KD-tree* is not complicated, the loss rate is relatively high. This is because the data points of the *KD-tree* are used as the point of separation of the nodes and spaces of the book at the same time, and the retrieval directly using the *KD-tree* cannot guarantee that all the correct candidate sets are selected. Therefore, we separate the storage of the segmentation line and the spatial information, and combine the *R-tree* segmentation strategy with the filter optimization algorithm [41] while ensuring performance and efficiency.

References

[1] J. Assfalg *et al.*, "Probabilistic Similarity Search for Uncertain Time Series", in *Proc. of the International Conference on Scientific and Statistical Database Management*, 2009, pp. 435–443. https://doi.org/10.1007/978-3-642-02279-1_31

[2] M.-Y. Yeh *et al.*, "PROUD: A Probabilistic Approach to Processing Similarity Queries over Uncertain Data Streams", in *Proc. of the 12th International Conference on Extending Database Technology*, 2009, pp. 684–695. http://dx.doi.org/10.1145/1516360.1516439

[3] S. R. Sarangi and K. Murthy, "'DUST: A Generalized Notion of Similarity Between Uncertain Time Series", in *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010, pp. 383–392. http://dx.doi.org/10.1145/1835804.1835854

[4] S. Hasegawa and T. Itoh, "Optimal Online Algorithms for the Multi-Objective Time Series Search Problem", *Theoretical Computer Science*, vol. 718, pp. 58–66, 2018. http://dx.doi.org/10.1016/j.tcs.2017.01.008

[5] G. E. A. P. A. Batista *et al.*, "A Complexity-Invariant Distance Measure for Time Series", in *Proc. of the SIAM International Conference on Data Mining*, 2011, pp. 699–710. http://dx.doi.org/10.1137/1.9781611972818.60

[6] C. C. Kuo *et al.*, "Time Series Index for GIS Partial Discharge Detection" in *Proc. of the Asia-Pacific International Conference on Lightning*, 2011, pp. 364–367. http://dx.doi.org/10.1109/APL.2011.6110143

[7] K. L. Liao *et al.*, "Wavelet Decomposition Algorithm for Uncertain Data Streams", in *Proc. of the International Conference on Computer Science & Education*, 2011, pp. 965–970. http://dx.doi.org/10.1109/ICCSE.2011.6028796

[8] D. Oliver *et al.*, "Geo-Referenced Time-Series Summarization Using k-Full Trees: A Summary of Results", in *Proc. of the 2012 IEEE International Conference on Data Mining Workshops*, 2012, pp. 797–804. http://dx.doi.org/10.1109/ICDMW.2012.64

[9] M. Orang and N. Shiri, "A Probabilistic Approach to Correlation Queries in Uncertain Time Series Data" in *Proc. of the 2012 ACM International Conference on Information and Knowledge Management*, 2012, pp. 2229–2233. http://dx.doi.org/10.1145/2396761.2398607

[10] J.-W. Roh *et al.*, "Efficient Bitmap-Based Indexing of Time-Based Interval Sequences", *Information Sciences*, vol. 194, pp. 38–56, 2012. http://dx.doi.org/10.1016/j.ins.2011.08.013

[11] Y. Zuo *et al.*, "Similarity Matching over Uncertain Time Series", in *Proc. of the International Conference on Computational Intelligence & Security*, 2012, pp. 1357–1361. http://dx.doi.org/10.1109/CIS.2011.302

[12] M. Orang and N. Shiri, "An Experimental Evaluation of Similarity Measures for Uncertain Time Series", in *Proc. of the 18th International Conference on Database Engineering & Applications Symposium*, pp. 261–264, 2014. http://dx.doi.org/10.1145/2628194.2628207

[13] X. Lian *et al.*, "Pattern Matching Over Cloaked Time Series", in *Proc. of the IEEE 24th International Conference on Data Engineering*, 2008, pp. 1462–1464. http://dx.doi.org/10.1109/ICDE.2008.4497590

[14] M. S. Gil *et al.*, "Fast Index Construction for Distortion-Free Subsequence Matching in Time-Series Databases" in *Proc. of the International Conference on Big Data and Smart Computing*, 2015, pp. 130–135. http://dx.doi.org/10.1109/35021BIGCOMP.2015.7072822

[15] C. Faloutsos *et al.*, "Fast Subsequence Matching in Time-Series Databases" in *Proc. of the 1994 ACM SIGMOD International Conference on Management of Data*, 1994, pp. 419–429. https://doi.org/10.1145/191839.191925

[16] U. Agarwal and A. S. Sabitha, "Time Series Forecasting of Stock Market Index", in *Proc. of the India International Conference on Information Processing*, 2016, pp. 1–6. http://dx.doi.org/10.1109/IICIP.2016.7975381

[17] M. Orang and N. Shiri, "Improving Performance of Similarity Measures for Uncertain Time Series Using Preprocess Techniques", in *Proc. of the 27th International Conference on Scientific and Statistical Database Management*, 2015, pp. 1–12. http://dx.doi.org/10.1145/2791347.2791385

[18] M. Orang and N. Shiri, "Correlation Analysis Techniques for Uncertain Time Series" *Knowledge and Information Systems*, vol. 50, no. 1, pp. 79–116, 2017. http://dx.doi.org/10.1007/s10115-016-0939-7

[19] B. Goswami *et al.*, "Abrupt Transitions in Time Series With Uncertainties", *Nature Communications*, vol. 9, no. 1, p. 48, 2018. http://dx.doi.org/10.1038/s41467-017-02456-6

[20] R. Cheng *et al.*, "Efficient Indexing Methods for Probabilistic Threshold Queries Over Uncertain Data" in *Proc. of the International Conference on Very Large Data Bases*, pp. 876–887, 2004. http://dx.doi.org/10.1016/B978-012088469-8.50077-2

[21] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching", in *Proc. of the 1984 ACM SIGMOD International Conference on Management of Data*, pp. 47–57, 1984. http://dx.doi.org/10.1145/602259.602266

[22] N. Roussopoulos and D. Leifker, "Direct Spatial Search on Pictorial Databases Using Packed R-Trees", *SIGMOD Record*, vol. 14, no. 4, pp. 17–31, 1985. http://dx.doi.org/10.1145/971699.318900

[23] U. Deppisch, "S-Tree: A Dynamic Balanced Signature Index for Office Retrieval", in *Proc. of the 9th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* , 1986, pp. 77–87. http://dx.doi.org/10.1145/253168.253189

[24] N. Beckmann *et al.*, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles", in *Proc. of the 1990 ACM SIGMOD International Conference on Management of Data*, 1990, pp. 322–331. http://dx.doi.org/10.1145/93597.98741

[25] B. K. Yi and C. Faloutsos, "Fast Time Sequence Indexing for Arbitrary Lp Norms", in *Proc. of the 26th International Conference on Very Large Data Bases*, 2000, pp. 385–394. http://dx.doi.org/10.1184/r1/6605618

[26] S. Berchtold *et al.*, "The x-Tree: An Index Structure for High-Dimensional Data", in *Proc. of the 22nd International Conference on Very Large Data Bases*, 1996, pp. 28–39.

[27] G. Kaiser, "The Fast Haar Transform", *IEEE Potentials*, vol. 17, no. 2, pp. 34–37, 1998. http://dx.doi.org/10.1109/45.666645

[28] D. M. Woodbridge *et al.*, "Time Series Discord Detection in Medical Data Using a Parallel Relational Database", in *Proc. of the 2015 IEEE International Conference on Bioinformatics and Biomedicine*, 2015, pp. 1420–1426. http://dx.doi.org/10.1109/BIBM.2015.7359885

[29] M. Allen *et al.*, "Real-Time In-Network Distribution System Monitoring to Improve Operational Efficiency", *Journal AWWA*, vol. 103, no. 7, pp. 63–75, 2011. https://doi.org/10.1002/j.1551-8833.2011.tb11495.x

[30] P. Zou *et al.*, "Fast Similarity Matching on Data Stream with Noise", in *Proc. of the 2008 IEEE 24th International Conference on Data Engineering Workshop*, 2008, pp. 194–199. http://dx.doi.org/10.1109/ICDE.1998.655778

[31] B. K. Yi *et al.*, "Efficient Retrieval of Similar Time Sequences Under Time Warping", in *Proc. of the 1998 International Conference on Data Engineering*, 1998, pp. 201–208. http://dx.doi.org/10.1109/ICDE.1998.655778

[32] W. Yi-Leh *et al.*, "A Comparison of DFT- and DWT-Based Similarity Search in Time-Series Databases", in *Proc. of the 9th International Conference on Information and Knowledge Management*, 2000, pp. 488–495. http://dx.doi.org/10.1145/354756.354857

[33] R. Agrawal *et al.*, "Efficient Similarity Search in Sequence Databases", in *Proc. of the 1993 Foundations of Data Organization and Algorithms*, pp. 69–84, 1993. https://doi.org/10.1007/3-540-57301-1_5

[34] S. Y. Park *et al.*, "Efficient Searches for Similar Subsequences of Different Lengths in Sequence Databases", in *Proc. of the 2000 International Conference on Data Engineering*, 2000, pp. 23–32. http://dx.doi.org/10.1109/ICDE.2000.839384

[35] D. Rafiei and A. Mendelzon, "Efficient Retrieval of Similar Time Sequences Using DFT", in *Proc. of the 1998 International Conference on Foundations of Data Organizations and Algorithms*, 249–257, 1998. https://arxiv.org/abs/cs/9809033

[36] R. Ma *et al.*, "Solar Flare Prediction Using Multivariate Time Series Decision Trees", in *Proc. of the 2017 IEEE International Conference on Big Data*, 2017, pp. 2569–2578. http://dx.doi.org/10.1109/BigData.2017.8258216

[37] R. Ma and R. A. Angryk, "Distance and Density Clustering for Time Series Data", in *Proc. of the 2017 IEEE International Conference on Data Mining Workshops*, 2017, pp. 25–32. http://dx.doi.org/10.1109/ICDMW.2017.11

[38] R. Ma *et al.*, "A Data-Driven Analysis of Interplanetary Coronal Mass Ejecta and Magnetic Flux Ropes", in *Proc. of the 2016 IEEE International Conference on Big Data*, 2016, pp. 3177–3186. http://dx.doi.org/10.1109/BigData.2016.7840973

[39] R. Ma *et al.*, "Coronal Mass Ejection Data Clustering and Visualization of Decision Trees", *The Astrophysical Journal Supplement Series*, vol. 236, no. 1, p. 4, 2018. http://dx.doi.org/10.3847/1538-4365/aab76f

[40] Y. Sakurai *et al.*, "Stream Monitoring under the Time Warping Distance", in Proc. of the *IEEE International Conference on Data Engineering*, pp. 1046–1055, 2007. http://dx.doi.org/10.1.1.417.2458

[41] D. W. Zheng *et al.*, "Spatial Index for Uncertain Time Series", *Journal of Computing and Information Technology*, vol. 26, no. 3, pp. 191–207, 2018. http://dx.doi.org/10.20532/cit.2018.1004248

Received: December 2018
Revised: February 2020
Accepted: April 2020

Contact addresses:
Ruizhe Ma
Georgia State University
Atlanta
USA
e-mail: cstnuaa@163.com

Diwei Zheng
Nanjing University of Aeronautics and Astronautics
Nanjing
China
e-mail: zheng@163.com

Li Yan*
Nanjing University of Aeronautics and Astronautics
Nanjing
China
e-mail: yanli@nuaa.edu.cn
*Corresponding author

Ruizhe Ma received her PhD degree from the Department of Computer Science at the Georgia State University, USA. Her research interests include time series analysis, Big Data processing and data mining.

Diwei Zheng received his Master degree from the College of Computer Science and Technology at the Nanjing University of Aeronautics and Astronautics, China. His research interests include time series analysis and uncertain data management.

Li Yan is a full professor at the College of Computer Science and Technology at the Nanjing University of Aeronautics and Astronautics, China. Her current research interests include uncertain data and knowledge engineering.