Efficient Similarity Search over Future Stream Time Series

Xiang Lian, *Student Member*, *IEEE*, and Lei Chen, *Member*, *IEEE*

Abstract—With the advance of hardware and communication technologies, stream time series is gaining ever-increasing attention due to its importance in many applications such as financial data processing, network monitoring, Web click-stream analysis, sensor data mining, and anomaly detection. For all of these applications, an efficient and effective similarity search over stream data is essential. Because of the unique characteristics of the stream, for example, data are frequently updated and real-time response is required, the previous approaches proposed for searching through archived data may not work in the stream scenarios. Especially, in the cases where data often arrive periodically for various reasons (for example, the communication congestion or batch processing), queries on such incomplete time series or even future time series may result in inaccuracy using traditional approaches. Therefore, in this paper, we propose three approaches, *polynomial*, *Discrete Fourier Transform (DFT)*, and *probabilistic*, to predict the unknown values that have not arrived at the system and answer similarity queries based on the predicted data. We also apply efficient indexes, that is, a multidimensional hash index and a B^+-tree, to facilitate the prediction and similarity search on future time series, respectively. Extensive experiments demonstrate the efficiency and effectiveness of our methods for prediction and answering queries.

Index Terms—Similarity search, stream time series, prediction, polynomial, DFT, probabilistic.

1 Introduction

Recently, stream time series data management has become a hot research topic due to its wide application usages such as Internet traffic analysis [8], sensor network monitoring [51], moving object search [11], financial data analysis [44], [49], and the like. All these applications require continuously monitoring stream time series data in real time. Compared to traditional archived time series data, stream time series data have their own characteristics, that is, data are frequently updated. Therefore, previous methods for similarity search over archived time series data may not work in this scenario. For example, in sensor networks, the system continuously receives data from sensors, which often arrive periodically due to various reasons such as communication congestion in the network or batch processing [2]. In this case, for similarity queries over the time when data are not being updated, previous approaches can only perform searches on incomplete time series (or previously received time series). Thus, the answer is often inaccurate due to the missing data. Furthermore, some other applications may even request a similarity search on the data some time in the future. As an example, in a real-world coal mine surveillance application [45], hundreds of sensors are deployed throughout channels to measure the density of oxygen, gas, and dust, together with the temperature, humidity, and structural integrity in the mine. For the safety of workers, the coal mine manager needs to detect as early as possible the gas leakage or low oxygen density, which follows certain patterns in a contour map. In this case, since we do not have any knowledge of the exact future subsequences (from sensor data), queries cannot be answered without guesswork. Therefore, it is very important to predict future subsequences (patterns) and efficiently perform a similarity search over them to find emergency events following particular patterns. Motivated by this, in this paper, we focus on tackling the problem of predicting values that have not arrived in the stream time series and building efficient indexes for both prediction and similarity search on these "future" series.

Previous work on predicting future data mainly use fuzzy methods [25], [32] or data mining techniques [18], [41], [43], [31] to extract features from the training data set and perform the prediction tasks on real-time series. However, in order to achieve accurate results that are insensitive to the evolving data, these methods usually require training the predictors on the actual data at high cost. Therefore, although these approaches can lead to good offline prediction accuracy given appropriate training, they are generally not suitable for the online stream environment, which requires low prediction and training costs. Moreover, previous methods often design complicated models that are specific to some concrete applications, for example, prediction in binary time series [17]. However, in this paper, we are seeking general solutions that are straightforward yet effective. Furthermore, previous approaches do not give any confidence for the prediction (the confidence can be arbitrary within [0, 1]), whereas one of our proposed methods, the *probabilistic* approach, can predict values while explicitly providing a confidence, which has many applications. As in the earlier example [45], predicting an emergency (for example, the dust density exceeds a threshold in a coal mine) often requires the prediction confidence.

The main contributions of our paper are summarized as follows:

• *The authors are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China. E-mail: {xlian, leichen}@cse.ust.hk.*

Manuscript received 29 Sept. 2006; revised 23 Apr. 2007; accepted 14 Aug. 2007; published online 4 Sept. 2007.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0458-0906. Digital Object Identifier no. 10.1109/TKDE.2007.190666.

1. We present the polynomial approach that predicts future values based on the approximated curve of the most recent values.
2. We illustrate the Discrete Fourier Transform (DFT)-based prediction technique that forecasts the future values using approximations in the frequency domain.
3. We propose another prediction method, the probabilistic approach, to predict future values according to aggregates of all the data in history. Moreover, our probabilistic approach can provide the confidence of predicting values and is adaptive to the possible change of data, because it can efficiently feed the prediction error back.
4. We present indexes for the probabilistic approach that efficiently facilitate prediction and similarity queries in the stream environment.
5. Last but not least, we further extend the probabilistic approach to the group probabilistic approach by utilizing the correlations among stream time series.

The rest of the paper is organized as follows: We present a brief review on the related work about the similarity search on the archived time series and prediction techniques in Section 2. Section 3 formally defines our problem and illustrates the general framework. We demonstrate our polynomial, DFT, and (group) probabilistic approaches in Section 4. Section 5 discusses indexes for the probabilistic method. We evaluate the prediction and query accuracy, as well as efficiency, with extensive experiments. Finally, in Section 7, we conclude this paper.

2 Related Work

Section 2.1 reviews the traditional similarity search in time-series databases, as well as various dimensionality reduction techniques. Section 2.2 briefly presents the previous work on value predictions.

2.1 Similarity Search in Time Series Databases

Similarity search over time series data has been studied for many years. Most of the proposed methods focus on searching over historical data. Specifically, given a query sequence $Q\ (= \{q_1, q_2, \ldots, q_L\})$ of length L and similarity threshold ε, a *similarity query* finds all the (sub)sequences $S\ (= \{s_1, s_2, \ldots, s_L\})$ with length L in the database that are similar to Q, satisfying $dist(Q,S) \le \varepsilon$, where the distance function $dist(Q,S)$ between Q and S can be L_p-norms [1], [13], *Dynamic Time Warping* (DTW) [3], [23], [26], [47], *Longest Common Subsequence* (LCSS) [5], [12], [40], *Edit Distance on Real Sequence* (EDR) [11], *Edit Distance with Real Penalty* (ERP) [9], or some other measures. For simplicity, throughout this paper, we use the *euclidean distance* (L_2-norm) between Q and S as our metric, which is defined as follows:

$$dist(Q,S) = \sqrt{\sum_{i=1}^{L} (q_i - s_i)^2}. \quad (1)$$

In order to efficiently perform the similarity search, Agrawal et al. [1] and Faloutsos et al. [13] transform the (sub)sequences of length L to lower s-dimensional points ($s \ll L$) with a dimensionality reduction technique, DFT [1], and then insert them into an R-tree [4]. Other reduction techniques include *Singular Value Decomposition* (SVD) [21], *Discrete Wavelet Transform* (DWT) [33], *Piecewise Aggregate Approximation* (PAA) [46], *Adaptive Piecewise Constant Approximation* (APCA) [22], and *Chebyshev Polynomials* (CP) [10]. All of these techniques follow the *lower bounding lemma*, that is, the distance between any two converted s-dimensional points is never greater than that between the original points in the L-dimensional space. This crucial feature can guarantee no *false dismissals* during the similarity search in the reduced space. In particular, the similarity query can be answered by issuing a normal *range query* over the R-tree index [4] and filtering out *false alarms* in the retrieved candidate set. Other queries such as *k nearest neighbor* (kNN) queries [37] can be easily extended by applying the similar filter-refinement framework.

Some previous methods for the similarity search assume knowing the underlying data model of time series in advance. For example, Kalpakis et al. [20] assume that the time series follow the *Auto-Regressive Integrated Moving Average* (ARIMA) models, which can be represented by a few *autoregression coefficients*. Kalpakis et al. defined a similarity measure between two time series using the *euclidean distance* between their *cepstral coefficients* derived from *autoregression coefficients* and applied it to cluster ARIMA time series. Other data models include the Markov model [34], the *autoregressive tree* (ART) model [31], and so on.

Gao and Wang [15] propose a framework for answering similarity queries on future data. In particular, they apply DFT reduction on the predicted data to find cross correlations of time series before the actual data arrive and perform either *nearest neighbor* or *range query* after the actual data income, considering the prediction error between the actual and predicted values. However, the authors only present a general framework, in which any prediction method can be used, and they did not propose any specific prediction method. In their experiments, synthetic prediction errors (that is, square root, linear, and square errors) are used. Gao et al. [14] also present a disk-based framework to answer *continuous nearest neighbor queries* via prefetching, which applies a simple prediction method by estimating the value with the actual values at the previous time stamp.

In summary, previous work on similarity search either are based on complete archived time series or provide a framework for preparing (predicted) data before the actual data arrive and answering queries after the data's arrival. Our work, however, focuses on effectively predicting the (incomplete) future time series in a streaming scenario over which an accurate query result can be obtained even before the actual data arrive.

2.2 Prediction Techniques

Related work on predicting values involve techniques that apply fuzzy rules [25], [32] and data mining approaches [18], [41], [43]. The problem of predicting unknown values is defined as follows: Assume we know H consecutive values, $x_1, x_2, \ldots$, and x_H, and want to predict the next Δt future values $x_{H+1}, x_{H+2}, \ldots$, and $x_{H+\Delta t}$. The traditional fuzzy predictor predicts values directly on the raw time series, where the fuzzy rules are in the form "if $x_1 = value_1, x_2 = value_2, \ldots, x_H = value_H$, then $x_k = value_k$, $k \in [H+1, H+\Delta t]$." However, this approach can guarantee the prediction accuracy only if the time series' statistics are stationary. Otherwise, with a dynamic series, it might fail to give accurate results. Kim and Lee [25] proposed a new predictor that is based on rules with differences of consecutive values, that is, if $x_1 - x_2 = value_1$, $x_2 - x_3 = value_2, \ldots, x_{H-1} - x_H = value_{H-1}$, then $x_{k-1} - x_k = value_{k-1}$, where $k \in [H+1, H+\Delta t]$." Policker and Geva [32] use the existing *fuzzy clustering algorithm*, the unsupervised optimal fuzzy clustering algorithm (UOFC) in [16], and the deterministic annealing approach in [36] to classify a large set of time series such that the series in each cluster are close and their temporal probabilistic behavior is similar. As a second step, the proposed algorithm predicts the future data from a mixture *probability distribution function* (PDF). In particular, for each future value, the fuzzy membership of each cluster is calculated, and the results are combined to produce an estimate of this future value.

For data mining techniques, Gestel et al. [18] make use of *Least Squares Support Vector Machines* (LS-SVMs) within the Bayesian evidence framework [30] to infer nonlinear models of financial time series, where a least squares error and equality constraints (instead of the inequality constraints) are applied. After training the prediction model, the predictor not only predicts the future financial time series but also tells the associated prediction risk for people to make optimal investment decisions. Wang et al. [43] propose the *wavelet packet multilayer perceptron* (WP-MLP) for prediction, based on the work of the *wavelet multilayer perceptron* (W-MLP) by Zhang and Benveniste [48]. Meek et al. [31] construct an ART over the training data set, which is a decision tree with an *autoregressive* (AR) model. During the prediction, they first traverse the tree from the root to the leaf node and then use the AR model stored in the leaf node to predict future values of time series. On the other hand, instead of predicting frequently appearing patterns in the future, Vilalta and Ma [41] predict the infrequent (*rare*) *target events* in a database containing event sequences. Vilalta and Ma assume that these *target events* are highly infrequent yet always having at least one very frequent event preceding themselves. Therefore, they apply the association rule mining techniques to retrieve such frequent event sets occurring before the target events and construct a rule-based model to predict the infrequent events.

These approaches, for example, clustering or training the neural network, however, incur a very high update cost for either mining fuzzy rules or training parameters in different models. Therefore, they are not applicable to efficient online processing in the stream environment, which requires low prediction and training costs. In this paper, we propose three approaches that can exactly achieve the requirement of the prediction efficiency and accuracy in stream time series.

3 Problem Definition and Framework

In this section, we formally define the similarity search in the future time series problem and present our two goals.

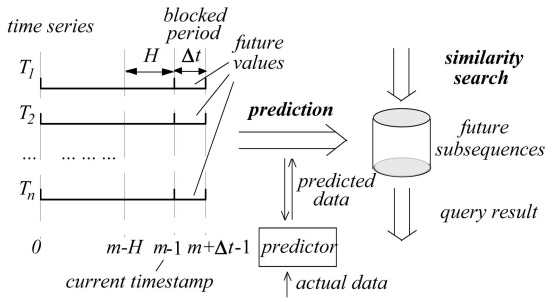


Fig. 1. The framework for our solution.

Fig. 1 illustrates the general framework, which mainly focuses on two tasks, that is, 1) predicting future values for each time series, and 2) answering similarity queries on subsequences in the future. Assume that we have n stream time series $T_1, T_2, \ldots,$ and T_n in the stream environment, each containing m ordered values at the current time stamp $(m-1)$, that is, $T_i = \{t_{i0}, t_{i1}, \ldots, t_{i(m-1)}\}$, where t_{ij} is the value at time stamp j in T_i. Due to various reasons, for example, the batch processing [2], these n stream time series would receive data periodically for every Δt time stamps. In other words, for each time series T_i, the future values $t_{im}, t_{i(m+1)}, \ldots,$ and $t_{i(m+\Delta t-1)}$ corresponding to time stamps m, $(m+1), \ldots,$ and $(m+\Delta t-1)$, respectively, arrive in a block fashion at the same time stamp $(m+\Delta t)$. Therefore, during the period from time stamp m to $(m+\Delta t-1)$, called the *blocked period*, the system knows nothing about Δt future values in each series. In order to answer similarity queries on future subsequences, we have to use a predictor to predict these unknown values based on the most recent H values and extract subsequences from them. After the actual data arrive at the system, the predictor is trained correspondingly.

The first goal of our work is to efficiently predict the $n \cdot \Delta t$ future values for n time series so that the prediction error is as low as possible. In particular, we define the *prediction error* as the squared *euclidean distance* (defined in (1)) between the predicted and actual values divided by the maximum possible prediction error. That is, for time series T_i, the *prediction error* $Err_{pred}(T_i)$ is measured by

$$Err_{pred}(T_i) = \sum_{j=m}^{m+\Delta t-1} (t_{ij} - t_{ij}{}')^2 / E_{\max}, \tag{2}$$

where t'_{ij} corresponds to the predicted value of t_{ij}. Note that in (2), $E_{\max}$ is the maximum possible error between the actual and predicted series (defined as $\Delta t \cdot (max\text{-}min)^2$, where each value t_{ij} in T_i are within $[min, max]$ [6], [19]), and $Err_{pred}(T_i)$ is a *relative error*.

We denote as $T_i[j:k]$ the subsequence $\{t_{ij}, t_{i(j+1)}, \ldots, t_{ik}\}$ of T_i, where $j \le k$. Those subsequences that contain at least one future value are called *future subsequences*. Specifically, $T_i[j:k]$ is a future subsequence if and only if $m \le k \le (m+\Delta t-1)$ holds. Given a query sequence Q of length L, the similarity search in the future time series is to retrieve all future subsequences $T_i[k-L+1:k]$ with length L from T_i such that $dist(T_i[k-L+1:k], Q) \le \varepsilon$, where $m \le k \le (m+\Delta t-1)$. Therefore, our second goal is to extract all future subsequences from n time series with

TABLE 1
Symbols and Their Descriptions

Symbols	Descriptions
T_i	the i-th time series
n	the number of time series
m	the number of data in series at present
L	the length of the query Q
ε	the similarity threshold
H	the number of historical data used to predict
Δt	the number of predicted values
K	the cardinality of the alphabet table
s_i	the symbol in the alphabet table
l	the length of each segment in PAA
h	the height of the trie ($h = H / l$)

predicted values and then efficiently answer similarity queries on them so that the query accuracy is as high as possible. In particular, the query accuracy is measured by the recall ratio of the query result:

$$recall\_ratio = \frac{recall\_num}{act\_num}, \tag{3}$$

where *recall_num* is the number of candidates in the query result that indeed match with query sequence Q, and *act_num* is the actual number of future subsequences that are in ε-match with Q. Table 1 summarizes the commonly used symbols in this paper.

4 Three Prediction Approaches

In this section, we present three approaches, *polynomial*, *DFT*, and *probabilistic*, to predict the future values of stream time series.

4.1 Polynomial Prediction

As mentioned before, we are interested in prediction techniques that are suitable for the stream environment, which requires low prediction and training costs. One approach that satisfies this requirement is to 1) use some commonly used curves to approximate the historical data and 2) predict future values according to curves.

Here, we choose the *polynomial* curve, since it is simple and efficient for online processing. In particular, among the family of polynomial curves, we focus on two types, *linear* and *quadratic* [35]. Fig. 2 illustrates an example of predicting the time series with *linear* and *quadratic* prediction methods. Following the convention [6], [19], we assume that all the values in time series T_i have a domain $[min, max]$.

Assume that we use H values $x_1, x_2, \ldots,$ and x_H to predict Δt consecutive values $x_{H+1}, x_{H+2}, \ldots,$ and $x_{H+\Delta t}$ in the future. Without loss of generality, let x_1 be the value at time stamp 1, x_2 at time stamp 2, and so on. We first consider the *linear prediction*, which assumes that all the $(H + \Delta t)$ values can be approximated by a single line in the form $x = a \cdot t + b$, where t is the time stamp, x is the estimated value, and parameters a and b characterize these $(H + \Delta t)$ data. Therefore, the predicted Δt values with a linear predictor are $(H + 1) \cdot a + b, (H + 2) \cdot a + b, \ldots,$ and $(H + \Delta t) \cdot a + b$, corresponding to $x_{H+1}, x_{H+2}, \ldots,$ and $x_{H+\Delta t}$, respectively.

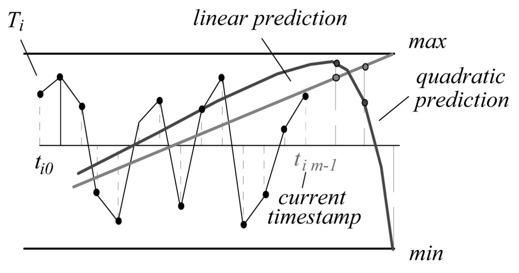


Fig. 2. Example of polynomial prediction.

We measure the *approximation error* of the linear curve on historical data by the squared *euclidean distance* between the actual series and the estimated one. That is,

$$Error_{linear\_appr} = \sum\nolimits_{i=1,\ldots,H} (a \cdot i + b - x_i)^2. \tag{4}$$

In order to obtain a good approximation, we aim at finding appropriate coefficients a and b such that the error $Error_{linear\_appr}$ is minimized. This can be achieved by the differential of $Error_{linear\_appr}$. Specifically, a and b must satisfy the following conditions: 1) $\partial Error_{linear\_appr}/\partial a = 0$ and 2) $\partial Error_{linear\_appr}/\partial b = 0$. As a result, we have

$$\begin{cases} a = 12 \cdot \sum\limits_{i=1}^{H} (i - (H+1)/2) \cdot x_i \Big/ H(H+1)(H-1), \\ b = 6 \cdot \sum\limits_{i=1}^{H} (i - (2H+1)/3) \cdot x_i \Big/ H(1-H). \end{cases} \tag{5}$$

Similarly, for the *quadratic prediction*, we approximate values by a quadratic curve in the form $x = a \cdot t^2 + b \cdot t + c$, where a, b, and c are parameters that characterize the data. The approximation error of a quadratic curve is measured by

$$Error_{quad\_appr} = \sum\nolimits_{i=1,\ldots,H} (a \cdot i^2 + b \cdot i + c - x_i)^2. \tag{6}$$

In order to minimize the error, three conditions must hold: 1) $\partial Error_{quad\_appr}/\partial a = 0$, 2) $\partial Error_{quad\_appr}/\partial b = 0$, and 3) $\partial Error_{quad\_appr}/\partial c = 0$. Thus, we can obtain coefficients a, b, and c by solving these equations with *Cram*'s rule.

As in Fig. 2, when the approximating curve (either *linear* or *quadratic*) intersects with the lower/upper bound min/max [6], [19], the predicted values after this time stamp would become meaningless. Furthermore, if the number of predicted values that are meaningful is not greater than Δt, curves of higher orders may have to be used. Nevertheless, the computational cost is high and, moreover, the prediction accuracy may not even be as good as low-order curves. Therefore, for simplicity, when the predicted value is below min (above max), we just treat the value as $min(max)$.

Another important issue is how to choose the value of H. Since stream data are changing all the time, we can adapt the value of H to such changes. In particular, we build up histograms for angles of any two consecutive values on historical data. Obviously, when the variance of the histogram is high, indicating the irregularity of the underlying data, a smaller value of H can achieve higher accuracy. Thus, as indicated by the histogram, we can select the value of H to better adapt to the stream data. Since the cost of either creating or maintaining the histogram is linear, it is especially suitable for tuning H during the online stream

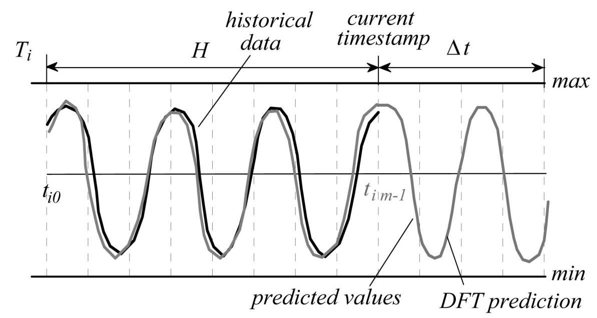


Fig. 3. Example of DFT prediction.

processing. In this way, our method can be adaptive to the change of data.

4.2 DFT Prediction

The *polynomial* approach above approximates the most recent data with a polynomial curve and predicts the unknown data with the curve, where the prediction process is based on the *time domain*. In the sequel, we propose the second method, *DFT prediction*, which approximates the historical data in a different domain, that is, the *frequency domain*.

Specifically, as illustrated in Fig. 3, assume that we use H historical values $x_1, x_2, \ldots,$ and x_H to predict Δt consecutive values $x_{H+1}, x_{H+2}, \ldots,$ and $x_{H+\Delta t}$ in the future. The *DFT prediction* first obtains H DFT coefficients [1] $x'_1, x'_2, \ldots,$ and x'_H of the most recent H historical data $x_1, x_2, \ldots,$ and x_H and then uses these H coefficients to reconstruct $(H + \Delta t)$ values, say, $y_1, y_2, \ldots, y_{H+\Delta t}$. The Δt reconstructed data $y_{H+1}, y_{H+2}, \ldots,$ and $y_{H+\Delta t}$ are considered as the predicted values of $x_{H+1}, x_{H+2}, \ldots,$ and $x_{H+\Delta t}$, respectively. Formally speaking, the resulting H DFT coefficients are formulated as follows:

$$x'_f = \frac{1}{\sqrt{H}} \sum_{t=1}^{H} x_t \cdot \exp(-j \cdot 2\pi \ (f-1)(t-1)/H) \quad f = 1, 2, \ldots, H, \tag{7}$$

$$y_t = \frac{1}{\sqrt{H+\Delta t}} \sum_{f=1}^{H} x'_f \cdot \exp(j \cdot 2\pi (f-1)(t-1)/(H+\Delta t)) \quad t = 1, 2, \ldots, (H+\Delta t). \tag{8}$$

The rational of the *DFT prediction* is that the frequencies of the H historical data and $(H + \Delta t)$ values (including both H historical data and Δt future data) are expected to be quite similar. Therefore, in the frequency domain, the H DFT coefficients from historical data are also similar to the first H coefficients of $(H + \Delta t)$ values. We tested 24 *benchmark* data sets [23], [50], [9], which will also be used in our experimental study later, to verify this assumption, where these data sets cover a wide spectrum of applications and have different data characteristics. Specifically, we measure the *relative error* of H DFT coefficients from H historical data and $(H + \Delta t)$ values (including H historical data and Δt future data), that is, the euclidean distance between H DFT coefficients from H historical data and that from $(H + \Delta t)$ values divided by the sum of H square coefficients from $(H + \Delta t)$ values. Fig. 4 illustrates the experimental results, where $H = 512$, and $\Delta t = 256$. We can see that errors are quite small, that is, most of them are below 15 percent, which confirms our assumption.

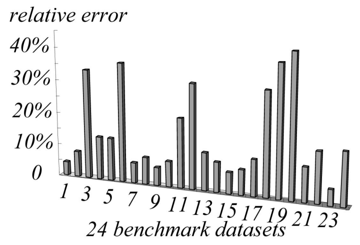


Fig. 4. Relative error of DFT coefficients.

Therefore, if data remain approximately the same in the frequency domain on historical and future data, it is very likely that the *DFT* method will offer an accurate result. As mentioned in the *polynomial* approach (Section 4.1), given the fixed system parameter Δt, the value of H can be dynamically adjusted by building up a histogram that contains the frequency of different angles from historical data at consecutive time stamps and detecting the change of data. Furthermore, the *DFT prediction* approach has a very nice feature that DFT coefficients can be *incrementally* computed by the work of Kontaki and Papadopoulos [27]. In particular, it requires only $O(1)$ cost to update each DFT coefficient upon the arrival of new data. Thus, this method is efficient for online processing.

4.3 Probabilistic Prediction

Our third prediction approach is based on the observation that those subsequences that appear frequent in history have a higher probability of occurring again in the future. Motivated by this, we utilize statistics on the entire historical data, rather than a few most recent in the *polynomial* or *DFT* solution, to predict future values. Specifically, we propose our *probabilistic* approach, which extracts the symbolic representation of historical subsequences, as well as their aggregate information, to predict the future symbols with probabilities based on aggregates that summarize the entire historical data and finally output future subsequences for similarity search.

Without loss of generality, we consider the prediction problem on a single time series T, which can be easily extended to the case of $n(> 1)$ time series. The details of the approach are listed as follows: Assume that at the current time stamp, we know the most recent H historical data and have to predict the future Δt values. In order to store the statistics in history, we maintain a structure, called the *aggregate trie*. Specifically, for each historical subsequence of length H, we update the aggregate trie in four steps:

1. divide it into h disjoint segments with identical length l, that is, $H = h \cdot l$,
2. obtain the mean value avg_i for each segment i, where $1 \leq i \leq h$,

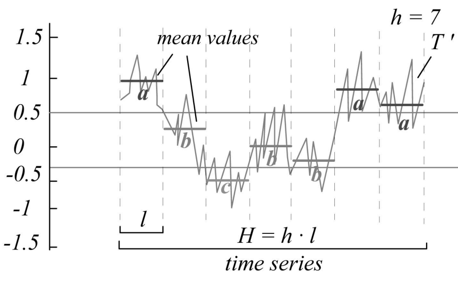


Fig. 5. The discrete version of a time series.

3. convert each avg_i into a symbol s_i, that is, transforming the subsequence to its symbolic representation $s_1s_2 \ldots s_h$, and
4. insert the string $s_1s_2 \ldots s_h$ into the aggregate trie with height h.

The first three steps transform the time series to its symbolic representation as proposed by Lin et al. [28] and Keogh et al. [24]. Fig. 5 illustrates an example of this transformation. Assume that the value domain of a time series T' is [-1.5, 1.5]. We partition it into three smaller ranges of equal size, say, [0.5, 1.5], [-0.5, 0.5), and [-1.5, -0.5), which correspond to three symbols a, b, and c, respectively. Note that in case a priori knowledge of stream data is known, one can divide the value domain into small ranges of different lengths [28], [24]. As a second step, we divide the time series T' of length H into seven segments of equal length, take the average value avg_i within each segment i, and finally convert each avg_i into a unique symbol. For example, the mean value avg_1 in the first segment falls into the range [0.5, 1.5], so we map it to the symbol "a." After the discretization, T' is represented by a string consisting of ordered symbols (also called SAX representation). Specifically, the discrete version of T' is the string "$abcbbaa$." One of the advantages of discretizing the time series is its space efficiency. That is, if there are in total K symbols in the alphabet table, each sequence requires only $h \cdot \lceil log_2(K) \rceil$ bits at most, where $h = H/l$. Note that although the symbolic representation is only an approximation of the time series, as demonstrated in our experimental section later, it can be applied to predict the future values effectively.

The fourth step inserts all the time series strings into an aggregate trie. However, in contrast to an ordinary trie, each node entry in our aggregate trie contains a triple $< freq, hit, miss >$, where $freq$ is the frequency that a string (from the root to leaf) appears in the time series, hit is the times that our prediction succeeds, and $miss$ is that it fails. Intuitively, if the frequency $freq$ of a particular string is high, then it will have a higher probability of appearing in the future. On the other hand, if our prediction of a certain string fails quite often, that is, its aggregate $miss$ is large, then we have to lower the chance of choosing this string as the prediction result. In other words, our prediction error can be fed back by such aggregates.

In the sequel, we use Fig. 6 as an example to illustrate our *probabilistic* approach. Assume that $h = 4$, $l = 2$, and $\Delta t = 6$ (that is, $H = 8$). Initially, the aggregates in each node of the trie are in the form $< 0, 0, 0 >$. Let the current time stamp be 7. The historical subsequence

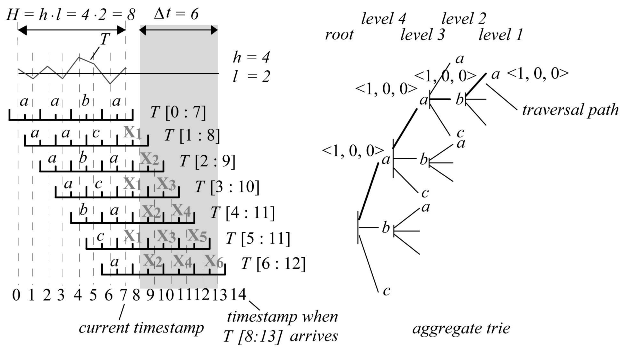


Fig. 6. Probabilistic prediction with aggregate trie.

$T[0:7]$ of length 8 is first transformed to a string "$aaba$." Then, we insert it into the trie through the path "$aaba$." Meanwhile, increase by 1 the frequency $freq$ of each entry on the path in the form $< freq, hit, miss >$, that is, updating $< 0, 0, 0 >$ to $< 1, 0, 0 >$ in the example.

Next, we illustrate the process of predictions using the example in Fig. 6. In particular, we want to predict symbols $\mathrm{X}_1, \mathrm{X}_2, \ldots$, and X_6, based on the trie, corresponding to subsequences $T[7:8], T[8:9], \ldots$, and $T[12:13]$, respectively. For simplicity, we only consider predicting the symbol X_1. Since the symbolic representation of $T[1:8]$ is "$aac\mathrm{X}_1$," we traverse the trie from the root to entry "a" on level 4 and, then, "a" and "c" on levels 3 and 2, respectively. After we have reached "c" on level 2, we should decide which symbol is most likely to be X_1, that is, the procedure of the prediction. One intuitive idea is to select the next symbol with the highest frequency. For example, if the frequency of both symbols "a" and "c" is 0, whereas that of "b" is 2, then the string "$aacb$" has higher probability than "$aaca$" or "$aacc$" to appear in future. Therefore, we can choose b as X_1. Although this intuitive idea can predict the future based on historical data, it cannot quickly adapt to the change of data. In this paper, we predict future values based on both frequency and hit ratio. Since we also store hit and $miss$ information in each entry of the trie, our approach takes into account the feedback of the prediction error in a *probabilistic* way. In particular, for each symbol s_i with $< freq\_s_i, hit\_s_i, miss\_s_i >$, the probability $Prob\_s_i$ of selecting s_i as X_1 is proportional to $freq\_s_i \cdot hit\_s_i/(hit\_s_i + miss\_s_i)$. Moreover, in the string "$aac\mathrm{X}_1$," suffixes "ac" and "c" of "aac" can also be used to predict X_1 but in a less accurate way due to fewer known symbols. Therefore, we can assign decreasing weights to "aac," "ac," and "c" during predictions. In the sequel, however, we only consider predicting future symbols with the longest string, for example, "aac," for simplicity. Note that in some situations, for example, the $freq$ aggregates of all the possible symbols of X_1 is zero, we handle such exceptions by simply setting X_1 to its previous symbol in the string, that is, "c" in string "$acc\mathrm{X}_1$." Let $Prob_1$ be the probability of predicting X_1. Similarly, other symbols such as $\mathrm{X}_2, \mathrm{X}_3, \ldots$, and X_6 are predicted with probabilities $Prob_2, Prob_3, \ldots$, and $Prob_6$, respectively. This method has the advantage of telling users the confidence of our predicting a value, which is

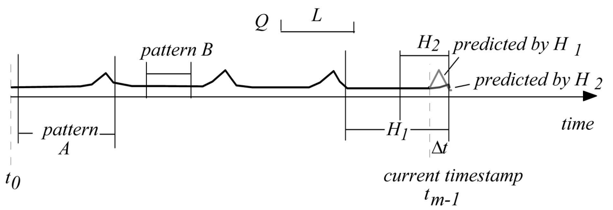


Fig. 7. Choosing appropriate values of H.

useful in many prediction applications, compared to the previous *polynomial* and *DFT* methods.

After all the unknown symbols have been predicted, future subsequences of length L are extracted without any difficulty. During the blocked period, whenever a query sequence Q of length L arrives, we can 1) perform the similarity search on these subsequences (in symbolic representation) and 2) output candidates whose lower bound distances from Q are not greater than the similarity threshold ε. Issues of indexing future subsequences are discussed in Section 5.

Finally, at time stamp 14, data $T[8], T[9], \ldots,$ and $T[13]$ will arrive in a batch. At this point, the actual symbols of $X_1, X_2, \ldots,$ and X_6 come out. As an example, for symbol X_1, if the prediction of X_1 is b, whereas the actual one is a (wrong prediction), then we feed the error back by increasing $miss$ in all entries $< freq, hit, miss >$ on the path of "$aacb$" by $Prob_1$ and $freq$ on that of "$aaca$" by 1. Otherwise, if the actual symbol of X_1 is indeed b (correct prediction), then in all entries on path "$aacb$," we increase $freq$ by 1 and hit by $Prob_1$. For other symbols, the procedure of updating aggregates in the trie is similar. Here, $Prob_1$ is the confidence that a symbol is predicted. If the symbol is wrongly predicted, we feed back this confidence by adding it to aggregate $miss$ as a penalty. The higher the confidence is, the heavier the penalty is in the case of failure. Similarly, with correct prediction, $Prob_1$ is added to aggregate hit as a bonus. As a result, since we feed back the prediction error with a penalty/bonus, the predicted error is expected to be small, which will be confirmed in our experimental section. Note that, however, for "acX_1X_3" of $T[3:10]$, when we update either a hit or a $miss$ aggregate, we always increase its value by $Prob_1 \cdot Prob_3$, since the prediction of X_3 is based on the predicted result of X_1.

4.3.1 Selection of H

Fig. 7 illustrates an example of prediction with two different values of H, H_1, and H_2. If we build a trie with the large value H_1 of an H, pattern A may appear very frequently, thus having more chance to predict the shape of pattern A (for example, a sudden burst in Fig. 7) in the future time stamps. On the other hand, for a smaller value H_2 of H, the trie we construct will contain pattern B with high frequency, so future values are more likely to be smooth. Now, the problem of how to choose an appropriate value of H arises. Our solution is to ensure the accuracy of user-specified queries. That is, if the user specifies the query Q of fixed length L, the value of H should be about the same value as L. In the previous example, the length L of Q is close to H_1, that is, Q has a higher probability to capture a sharp curve. Therefore, we choose H_1 to build the trie. In the case of various query lengths, we suggest using different values of H, for example, $H_{min}, 2 \cdot H_{min}, \ldots,$ and $2^{r-1} \cdot H_{min}$, that are exponentially increasing, where H_{min} is the minimum possible query length, and $2^{r-1} \cdot H_{min}$ is the maximum. When a query of length L arrives, we select the trie with $2^i \cdot H_{min}$ such that $2^{i-1} \cdot H_{min} \le (L + \Delta t) \le 2^i \cdot H_{min}$. In this way, our approach can achieve good query accuracy that is adaptive to the query length L.

4.4 Group Probabilistic Prediction

The *probabilistic* approach predicts the future values for each stream time series individually, based on the historical data, as well as previous prediction feedbacks. In this section, we study a special case where a number of stream time series appear to have correlations with each other. Formally, given two subsequences $R_1 = \{r_{11}, r_{12}, \ldots, r_{1w}\}$ and $R_2 = \{r_{21}, r_{22}, \ldots, r_{2w}\}$ of length w with means μ_1 and μ_2, respectively, the correlation coefficient $corr(R_1, R_2)$ is given as follows [49]:

$$corr(R_1, R_2) = \frac{\frac{1}{w}\sum_{i=1}^{w} r_{1i} r_{2i} - \mu_1 \mu_2}{\sqrt{\sum_{i=1}^{w} (r_{1i} - \mu_1)^2} \sqrt{\sum_{i=1}^{w} (r_{2i} - \mu_2)^2}}.$$

In many real applications [49], stream time series are often correlated. For example, some stock price time series may have quite a similar trend, since they are affected by the same types of financial factors. Moreover, in sensor networks, sensor data like temperature or humidity are collected from various sites, and those data obtained from spatially close sites tend to follow similar patterns. Inspired by this fact, we propose a novel *group probabilistic* approach, which utilizes this correlation information among a group of stream time series to correct the results predicted by the *probabilistic* approach. Specifically, we illustrate our basic idea using a simple example. Assume that we have a group of three correlated stream time series, which have three strings $S_1 = abbc$, $S_2 = bcbc$, and $S_3 = abba$, where the first three symbols in the strings are known, and the last ones in the strings (that is, c, c, and a) are predicted by the *probabilistic* approach. Here, let symbol a represent interval [0, 1] and $b(c)$ represent interval (1, 2] ((2, 3]). We consider the last two symbols of each string, that is, bc, bc, and ba in strings S_1, S_2, and S_3, respectively. Assuming that the three strings are correlated, they thus tend to have similar patterns. However, the last two symbols, bc of S_1 and S_2, correspond to an increasing trend, whereas ba in S_3 has a (conflicting) decreasing trend. Obviously, since there are two strings having the increasing trend, compared to one with decreasing trend, it is very likely that S_3 is falsely predicted. Therefore, our *group probabilistic* approach corrects this confliction by letting the last symbol a of S_3 to c.

In particular, we implement our *group probabilistic* approach by maintaining an array of size

$$2K - 1 (= 2(K - 1) + 1),$$

where K is the total number of symbols in the alphabet. Specifically, $(K - 1)$ entries of the array correspond to increasing trends (for example, ab, bc, and ac in the previous example), $(K-1)$ other entries correspond to decreasing trends (for example, ba, cb, and ca), and the last one is used for the constant trend (for example, aa, bb, and cc). Note that ab and bc (ba and cb) correspond to the same entry because they have the same increasing (decreasing) trend. Initially, these entries are set to zero. Then, for each stream time series in the correlated group, we obtain its predicted trend and increase the value of its corresponding entry by 1. Finally, we choose the trend that has the highest counting value as the *major trend* and correct the prediction results for all the series in the group. In the case where multiple entries have the same highest counting values, we do not perform the correction operation (that is, the prediction results are the same as that of the *probabilistic* approach), since it is not known which one is the best trend. We make use of the trends represented by the last two symbols instead of more. This is because more symbols would result in many possible combinations of symbols (trends) and smaller counting value in the *major trend* entry. In the worst case, each entry has the value at most 1 (that is, many entries have the same highest counting value 1). As a consequence, the *group probabilistic* approach becomes less effective in correcting the symbols predicted by the *probabilistic* approach.

Now, we discuss how to apply this prediction method to the stream scenario. Since the *group probabilistic prediction* can achieve good prediction accuracy only if the group contains correlated time series, we need to detect the correlations among stream time series. In particular, Zhu and Shasha [49] proposed an efficient method to incrementally monitor the correlations among multiple stream series. We use this method to efficiently calculate correlations among series and periodically check and validate the group membership for stream time series during the stream processing. Within each group, we apply the *group probabilistic* approach to correct the results obtained from the *probabilistic* approach, whereas for those outliers (that is, the group size is smaller than 3), we would just apply the *probabilistic prediction* approach. In this way, our *group probabilistic* approach can perform the prediction for correlated groups adaptive to the stream time series data.

As will be indicated by our empirical study in Section 6.6, the *group probabilistic* approach can achieve higher prediction accuracy and recall ratio for similarity queries than the *probabilistic* one.

5 Indexes for the Probabilistic Approach

In previous sections, we only consider prediction and similarity search on a single stream time series. In such a scenario, since there is only one trie for a single time series and the number Δt of predicted symbols is relatively small, in most cases, both prediction and similarity search can be processed in memory. However, in the scenario of n stream time series, for example, thousands of time series, the problem becomes more complex. For one thing, it is very unlikely that the memory can retain n tries (built from n series, respectively) and $n \cdot \Delta t$ predicted symbols without any overflow. Once we decide to flush some tries or predicted symbols back to the disk, the I/O cost becomes the bottleneck of the online processing, especially when tries are of a great height. Another thing is that without indexes, the sequential scan might be the only way to

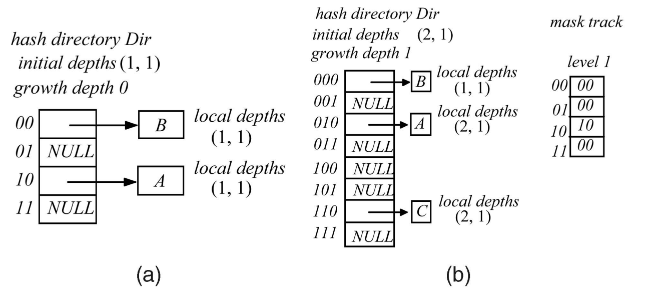


Fig. 8. Example of a 2D extendible hash. (a) Starting state. (b) Directory doubling.

perform the similarity search on $n \cdot \Delta t$ predicted subsequences but with very high cost.

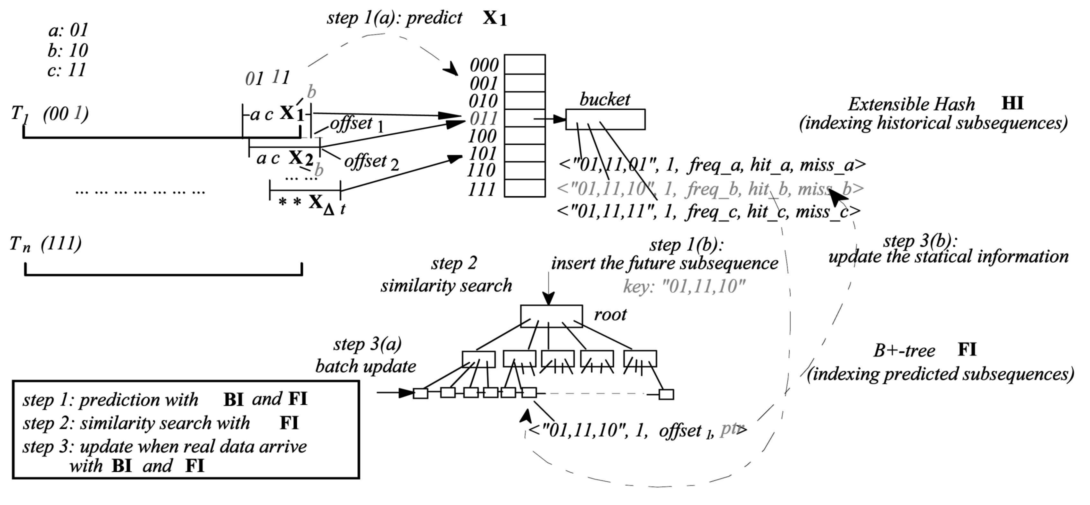
Motivated by this, we aim to build up efficient indexes to reduce the I/O cost. Section 5.1 discusses predictions, Section 5.2 illustrates the similarity search, and Section 5.3 presents the update method to aggregates (that is, error feedback).

5.1 The Alternative Index for Trie

As we have mentioned, it is not feasible to maintain n aggregate tries separately with great height h. Furthermore, we do not even need to materialize an actual trie. This is based on the observation that each path of the trie can be uniquely identified by a number of ordered integers (key). As an example, given the descending path "acb" of a trie, instead of traversing the trie from the root to leaf, we can alternatively retrieve entries with a multidimensional key corresponding to the path, for example, a unique key $< 1, 3, 2 >$ of "acb," assuming that a, b, and c are mapped to integers 1, 2, and 3, respectively.

Based on the above idea, we construct a single h-dimensional hash index **HI** for all the n time series. Here, we choose the hash index because of its good retrieval efficiency, compared to other multidimensional indexes such as R-tree [4]. That is, the cost of either searching or inserting one data in the hash structure is always $O(1)$. The only difference of our index from an ordinary hash such as the *extendible hash* is that keys of hash entries are multidimensional.

Previous work by Lin et al. [29] proposed a 3D *extendible hash* structure to organize the red, green, and blue (RGB) color data. Fig. 8 illustrates its basic idea with an example of a 2D extendible hash. Assume that each entry in the hash index contains a 2D integer key within the domain [0, 7] (for example, $< 010, 011 >$). As in Fig. 8a, the hash directory Dir initially contains two initial depths (1, 1), a *growth depth* 0, and the *global depth* 2 $(= 1 + 1 + 0)$. Now, suppose directory entries $Dir[10]$ and $Dir[00]$ point to buckets A and B, respectively. When a new data with key $< 110, 011 >$ arrives, we first extract the most significant bit from each dimension (the number of chosen bits depends on the initial depths), that is, 1 and 0, combine them together as the initial address "10," and insert data into bucket A pointed by $Dir[10]$. If bucket A overflows, however, we have to double the directory (see Fig. 8b) and increase the growth depth by 1. Moreover, we create a new bucket C that is pointed by $Dir[110]$. Then, bucket A is split along the dimension with the highest variance (for example, the first dimension). We distribute to bucket C those entries (for example, data with key $< 110, 011 >$) in A that have 1 as their second most



step 1(a): predict X1
a: 01
b: 10
c: 11
01 11
T1 (00 1)
a c X1
offset 1
a c X2
offset 2
* * XΔt
Tn (111)
000
001
010
011
100
101
110
111
bucket
Extensible Hash HI
(indexing historical subsequences)
<"01,11,01", 1, freq_a, hit_a, miss_a>
<"01,11,10", 1, freq_b, hit_b, miss_b>
<"01,11,11", 1, freq_c, hit_c, miss_c>
step 1(b): insert the future subsequence
key: "01,11,10"
step 3(b): update the statical information
step 2 similarity search
root
step 3(a) batch update
B+-tree FI
(indexing predicted subsequences)
<"01,11,10", 1, offset 1, ptr>
step 1: prediction with BI and FI
step 2: similarity search with FI
step 3: update when real data arrive with BI and FI

Fig. 9. Example of processes of predictions, queries, and updates.

significant bit along the first dimension. In addition, we keep track of the split history for future search in an array, called *mask track*, whose details are omitted here. When we search a specific data, the mask track is first looked up with its key level by level until the address of the bucket where the data possibly lie is obtained. Then, the data can be fetched by scanning entries in the bucket sequentially.

Although the work by Lin et al. [29] only used a 3D extendible hash structure, we can easily extend it to an index with arbitrary dimensions. By applying such a hash structure, we can index all the entries in the trie. Specifically, assuming that there are a total of K symbols in the alphabet table, we map them, for example, $s_1, s_2, \ldots,$ and s_K, to integers $1, 2, \ldots,$ and K, respectively. Any path "$s_{i1}s_{i2}\ldots s_{ih}$" of the trie for T_i can be transformed to h integers $i_1, i_2, \ldots$ and i_h. In particular, we extract the first $(h-1)$ integers $< i_1, i_2, \ldots, i_{h-1} >$ as the first $(h-1)$ ordered dimensions $< key_1, key_2, \ldots, key_{h-1} >$ in the hash entry, whereas the hth dimension key_h corresponds to the ID i of the time series T_i, for example, the ID 1 for time series T_1. Taking advantage of the fast retrieval in the hash index, we can either insert or search an entry very efficiently, with integers from "$s_{i1}s_{i2}\ldots s_{i(h-1)}$" and the series ID tid. Specifically, the initial hash address can be obtained as follows: We extract the most significant d_r bits of integer i_r corresponding to s_{ir} for all $1 \leq r \leq h-1$, as well as the least significant d_{tid} bits of tid in reverse order, concatenate these $(d_1 + d_2 + \ldots + d_{h-1} + d_{tid})$ bits to form the initial address of the hashing directory. The reason for our using the least significant bits of tid is its nice scalability in the hash index. In other words, when a new stream time series, for example, T_{n+1}, comes to the system, we assign to it with a greater ID $(n+1)$, which will not ruin the least significant bits of other series but will do to the most significant ones. As an example, assume that the total number of stream time series is originally three and only two bits are used to form addresses. When a new stream time series joins the system, it is assigned with ID 4 (with bit representation "100"). If we consider the most significant two bits, the stream time series with ID 3 (with bit representation "11" or "011") would change its two bits from the original "11" to "01," which leads to the costly update to hash index. On the other hand, if we consider the least significant two bits of the ID, its bits would not change, which is scalable to the total number of series.

In our *probabilistic* approach, each entry of the bucket in **HI** is in the form $< Hash(PA), tid, freq, hit, miss >$, where $Hash(PA)$ is a hashing function converting the path PA of length h in the trie into a single value, tid is the ID of the time series, and the last three attributes are aggregates the same as those stored in the trie. Note that here, the string "$s_{i1}s_{i2}\ldots s_{ih}$" on the path PA can be mapped to a value $Hash(PA)$ by concatenating all the bits of h integers $i_1, i_2, \ldots,$ and i_h from h symbols $s_{i1}, s_{i2}, \ldots,$ and s_{ih}, respectively. As an example, assume that there are three symbols, s_1, s_2, and s_3, in total and the height h of a trie is 4. The string "$s_1s_3s_2s_1$" can be converted into a value "01,11,10,01" in bit representation, and $Hash(\text{"}s_1s_3s_2s_1\text{"})$ is thus 121 in decimal representation. Note that given a hashed value $Hash(PA)$, we can also reversely obtain its original string "$s_1s_3s_2s_1$." Entries in the bucket are sorted in the order of tid and $Hash(PA)$ for the convenience of prediction. For the *group probabilistic* approach, the index **HI** is built in the same way as that of the *probabilistic* approach. After we get the predicted results from **HI**, the *group probabilistic* approach can be applied to improve the prediction accuracy.

5.2 The Index for Similarity Search

The second index we propose is a B$^+$-tree **FI**, which temporarily stores all the predicted future subsequences and facilitates the similarity search. In particular, the search key in **FI** is computed by converting h symbols of a predicted string into a single value, similar to the calculation of $Hash(PA)$ in **HI** mentioned before. Each entry of the leaf node is in the form $< Hash(PA), tid, offset, ptr >$, where $Hash(PA)$ is the key, tid is the ID of the time series, $offset$ is the end offset of the predicted subsequence, and ptr is a pointer pointing to the corresponding entry in **HI**.

During the blocked period, similarity queries are performed on **FI**. In particular, given a query Q of length L, our goal is to find candidate subsequences in the future such that their lower bound distances to Q are within ε. We illustrate the basic idea of our search procedure in an example in Fig. 9, where the B$^+$-tree **FI** has height $h' = 3$. Without loss of generality, assume that the *segment mean* representation $SM(Q)$ of a query series Q has three (that is, $h_q = L/l$)

segments corresponding to symbols q_1, q_2, and q_3, respectively. We start from the root of **FI**, considering the first $\lceil h_q/h' \rceil (= \lceil L/(h' \cdot l) \rceil = 1)$ symbol q_1 of $SM(Q)$. For each entry N_i in the root, if the minimum possible (squared) distance $LB\_dist_1^2$ from q_1 to (the first symbol of) any subsequence in N_i is smaller than ε^2, then we need to access the children of N_i; otherwise, N_i is ignored (since it cannot contain any query result). Note that since each symbol represents a value interval, the minimum distance between any two symbols is defined as the minimum possible distance between their corresponding intervals. We continue the running example by accessing a child N_i' of N_i on the second level of **FI**, considering the next $\lceil h_q/h' \rceil$ (that is, one) symbol q_2 of $SM(Q)$. This time, if the minimum possible (squared) distance $LB\_dist_2^2$ from q_2 to (the second symbol of) any subsequence in N_i' is smaller than $\varepsilon^2 - LB\_dist_1^2$, then we have to access its children; otherwise, N_i' can be safely pruned. Similarly, on the leaf level (that is, the last level of **FI**), we calculate the minimum (squared) distance $LB\_dist_3^2$ from q_3 to the third symbol of any subsequence S in the leaf node. If it holds that $LB\_dist_3^2 < \varepsilon^2 - LB\_dist_1^2 - LB\_dist_2^2$, then subsequence S is a candidate; otherwise, S is a *false alarm*. Finally, we use the actual values in Q to further refine the retrieved candidate set.

5.3 Combining All Together

By combining all the indexes together, we illustrate the entire process of the similarity search in future time series in an example in Fig. 9. Assume that the alphabet table is $\{a, b, c\}$ and the height h of the trie is 3. Let a be "01" in bit representation, b be "10," and c be "11." At the current time stamp, we consider predicting the future string "acX$_1$" for time series T_1. First (step 1(a)), we convert 3D vector $< a, c, 1 >$ into a single value "011" in bit representation by extracting the first bit from a ("01") and c ("11"), respectively, and the last bit "1" of tid 1. By searching "011" in **HI**, we find three entries

$$< Hash(\,''aca\,''), 1, freq\_a, hit\_a, miss\_a >,$$
$$< Hash(\,''acb\,''), 1, freq\_b, hit\_b, miss\_b >, \text{ and}$$
$$< Hash(\,''acc\,''), 1, freq\_c, hit\_c, miss\_c >$$

in a bucket, corresponding to strings "*aca*," "*acb*," and "*acc*," respectively. As discussed in the previous section, we compute the probability of selecting each entry as our prediction result and choose the one proportional to its probability. Without loss of generality, let the chosen symbol be b with probability $Prob_1$. Note that for the *group probabilistic* approach, after predicting values over a group of correlated stream time series, we can further correct the predicted symbols by considering the *major trend* in the group. Next (step 1(b)), we construct a B$^+$-tree **FI** for indexing future subsequences. In particular, we insert the entry $< Hash(\,''bca\,''), 1, offset_1, ptr >$ into **FI** with the key $Hash(\,''bca\,'')$ and pointer ptr pointing to $< Hash(\,''acb\,''), 1, freq\_b, hit\_b, miss\_b >$ in **HI**, where $offset_1$ is the end offset of the future subsequence. Predictions of other future subsequences are similar.

```
Algorithm Prob_Prediction
// step 1(a): prediction
1.  for each time series $T_i[0 : m - 1]$ $(1 \le i \le n)$
2.    for each subsequence $T_i[(m-1+j)-H+1:m-1+j]$ $(1 \le j \le \Delta t)$
3.      convert it into symbolic representation $s_{i1}s_{i2}\ldots s_{ih-1}X$
                                    // $X$ is the symbol to predict
4.      transform $s_{i1}$, $s_{i2}$, …, and $s_{i(h-1)}$ to integers $s_{i1}'$, $s_{i2}'$, …, and $s_{i(h-1)}'$
5.      search HI with keys $<s_{i1}', s_{i2}', \ldots, s_{i(h-1)}', i>$
6.      retrieve all entries in the form $<Hash("s_{i1}s_{i2}\ldots s_{i(h-1)}X_j"), i,$
            $freq\_X_j, hit\_X_j, miss\_X_j>$
7.      predict $X$ as $X'$ with probability $Prob(X') \propto freq\_X' \cdot hit\_$
            $X'/(hit\_X' + miss\_X')$
// step 1(b): construct the index for similarity search
8.        insert entry $<Hash("Xs_{i1}\, s_{i2}\ldots s_{i(h-1)}"), i, (m-1+j), ptr>$ into FI
              with the key $Hash("s_{i1}s_{i2}\ldots s_{i(h-1)}X")$ where $ptr$ points to
              $<Hash("s_{i1}s_{i2}\ldots s_{i(h-1)}X'"), i, freq\_X', hit\_X', miss\_X'>$
// step 2: answer similarity queries on FI
1.  Similarity_Search (FI, $Q$, $L$);
// step 3(a): after $\Delta t$ timestamps, batch update
1.  batch scan all the leaf nodes of FI
2.  for each entry $<Hash("Xs_{i1}\, s_{i2}\ldots s_{i(h-1)}"), i, (m - 1 + j), ptr>$
// step 3(b): update statistics
3.    obtain the true value $X''$ of $X$ in $T_i$ with end offset $(m-1+j)$
4.        obtain entry $<Hash("s_{i1}s_{i2}\ldots s_{i(h-1)}X'\ "), i, freq\_X', hit\_X',$
              $miss\_X'>$ in HI through $ptr$
5.    if $X' = X''$
6.      $freq\_X' = freq\_X' + 1$ and $hit\_X' = hit\_X' + Prob(X')$
7.    else $miss\_X' = miss\_X' + Prob(X')$
8.        retrieve entry $<Hash("s_{i1}s_{i2}\ldots s_{i(h-1)}X''\ "), i, freq\_X'', hit\_X'',$
              $miss\_X''>$
9.        set $freq\_X'' = freq\_X'' + 1$
10. recycle all the node in HI and repeat step 1 predicting the next
    $\Delta t$ values
End Prob_Prediction
```

Fig. 10. The pseudocode of probabilistic prediction.

During the blocked period, we always perform the similarity search on index **FI** (step 2). Then, after Δt time stamps, the actual data arrive at the system. We batch scan all the leaf nodes in **FI** (step 3(a)). For each entry we encounter, for example, $< Hash(\,''bca\,''), 1, offset_1, ptr >$, we obtain the true value of X$_1$ in the subsequence with end offset $offset_1$ in time series T_1. If X$_1$ is indeed b (as we predicted), access the entry

$$< Hash(\,''acb\,''), 1, freq\_b, hit\_b, miss\_b >$$

in **HI** through ptr and increase $freq\_b$ and hit_b by 1 and $Prob_1$, respectively; otherwise (wrong prediction, for example, X$_1 = a$), update the entry by increasing $miss\_b$ by $Prob_1$, retrieve the entry of

$$\textbf{HI} < Hash(\,''aca''), 1, freq\_a, hit\_a, miss\_a >$$

(insert a new one if it does not exist), and increment $freq\_a$ by 1 (step 3(b)). When all the leaf nodes of **FI** have been processed and aggregates in **HI** have been updated, we recycle all the nodes in **FI** and start a new round to predict values in the next Δt time stamps. Fig. 10 illustrates the pseudocode of the probabilistic algorithm.

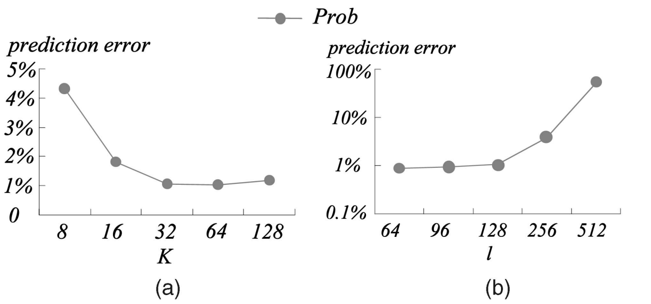


Fig. 11. Prediction error versus $K(l)$ (*periodic data set*). (a) Err_{pred} versus K. (b) Err_{pred} versus l.

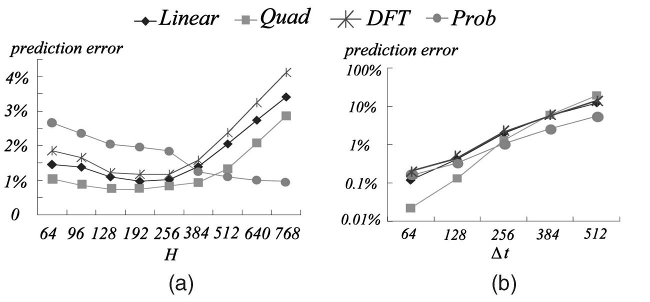


Fig. 12. Prediction error versus $H(\Delta t)$ (*periodic data set*). (a) Err_{pred} versus H. (b) Err_{pred} versus Δt.

6 Experimental Evaluation

In order to evaluate the effectiveness and efficiency of our proposed approaches, we run extensive experiments with both synthetic and real data sets. Specifically, synthetic data sets include the *periodic* data set (100,000, [http://kdd.ics.uci.edu/summary.data.type.html]) and the *randomwalk* data set (5,120 sequences of length 512) [22], [23], [50], whereas real data sets consist of 24 *benchmark* data sets (200 sequences of length 256 for each data set) [23], [50], [9] and the *sstock* data set (5,120 sequences of length 512) [42]. In particular, the real data sets, 24 *benchmark* data sets and the *sstock* data set, represent a wide spectrum of applications and data characteristics, which can be used to verify the universal usefulness of our approaches. Furthermore, when we perform our predictions, we always assume that the underlying data models are not known in advance. In order to verify the effectiveness, we compare the *probabilistic* prediction with *linear*, *quadratic*, and *DFT* predictions in terms of prediction and query accuracy under different parameters, that is, K, l, H, Δt, and ε. Section 6.1 demonstrates the experimental settings and tunes parameters K and l for the following experiments. Section 6.2 evaluates the prediction accuracy and efficiency of our probabilistic approach compared to that of *linear*, *quadratic*, *DFT*, *fuzzy*, and *ART* solutions. Section 6.3 presents the query accuracy of similarity search with the predicted future subsequences. In Sections 6.4 and 6.5, we present results of efficiency and scalability tests, respectively, for the similarity search. Section 6.6 illustrates the performance of the *group probabilistic* approach.

6.1 Experimental Settings and Parameter Tuning

The first set of experiments study the impact of parameters K and l on the prediction accuracy of our *probabilistic* approach and then fix their values for all the subsequent experiments, where K is the cardinality of the alphabet table, and l is the length of each segment in PAA. Note that, however, each predicted value in our proposed *probabilistic* approach is not a single value but a range corresponding to a symbol, for example, $[bound_r(i), bound_{r+1}(i)]$ at time stamp i. Let *act* be the actual value. We define the *prediction error* specific to our *probabilistic* approach, namely, the *probabilistic error* Err_{prob}, as follows:

$$Err_{prob}(act) = \sum_{i=m}^{m+\Delta t-1} ((bound_r(i) + bound_{r+1}(i)) \\ /2 - act)^2 / E_{\max}. \tag{9}$$

Intuitively, the *probabilistic error* Err_{prob} is the (squared) euclidean distance from the actual value to the middle value of the *predicted* range, that is, $(bound_r(i) + bound_{r+1}(i))/2$, divided by the maximum possible prediction error $E_{\max}$.

Fig. 11a illustrates the effect of parameter

$$K\ (= 8, 16, 32, 64, 128),$$

that is, the total number of symbols in the alphabet table, on the prediction accuracy with the *periodic* data set, where $l = 128$, $H = 512$, and $\Delta t = 256$. When K is small, that is, the value range of the time series is divided into only a few ranges, errors of the *probabilistic* method are high, since the granularity of the discretization is too coarse to give a precise result. When K is too large, however, errors are also high. This is mainly because aggregates in the hash index may not be well trained. Fig. 11b shows the effect of l on prediction errors on the *periodic* data set, where $l = 64, 96, 128, 256$, and 512, $H = 512$, and $\Delta t = 256$. Recall that given a fixed value of H, a large l indicates a small number of segments and coarse approximation of the historical data. Therefore, with the increase of l, the prediction error also increases dramatically. On the other hand, when l is too small, the resulting key concatenation of indexes may incur more cost. Thus, we can try different pair combinations of K and l values and empirically select a pair with a low *prediction error*, based on historical data. With similar results on other data sets, in the following experiments, we fix the value of K to 32 and l to 128.

6.2 Prediction Accuracy and Efficiency

After fixing values of K and l, we consider the effect of H and Δt on the prediction accuracy of four methods: *linear*, *quadratic*, *DFT*, and *probabilistic*. In particular, the *prediction accuracy* of *linear*, *quadratic*, and *DFT* methods is measured by the *prediction error* Err_{pred} defined in (2), whereas that of the *probabilistic* approach is measured by the *probabilistic error* Err_{prob} in (9). In Fig. 12a, we fix $\Delta t = 256$ and vary $H = 64, 96, 128, 196, 256, 384, 512, 640$ and 768. *Linear*, *quadratic*, and *DFT* predictions have errors that first decrease and then increase when H increases. This indicates that the most recent values have more importance in predicting future values, whereas too few historical data may result in a high prediction error. In contrast, the *probabilistic* approach can adapt to data change and give better result when more historical data are used (that is, H is large). Fig. 12b varies Δt from 64 to 512, where $H = 512$. All the

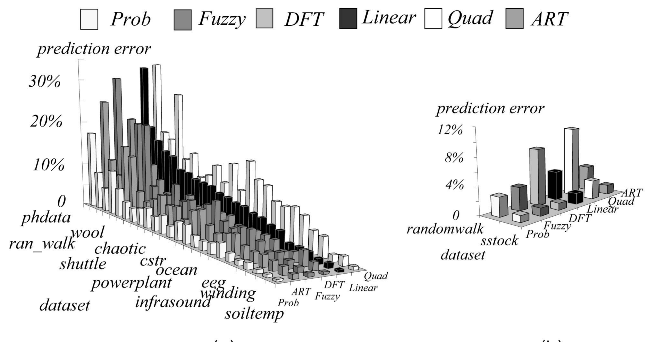


Fig. 13. Prediction accuracy versus data sets. (a) Twenty-four *benchmark* data sets (Err_{pred}). (b) *Sstock* and *randomwalk* (Err_{pred}).

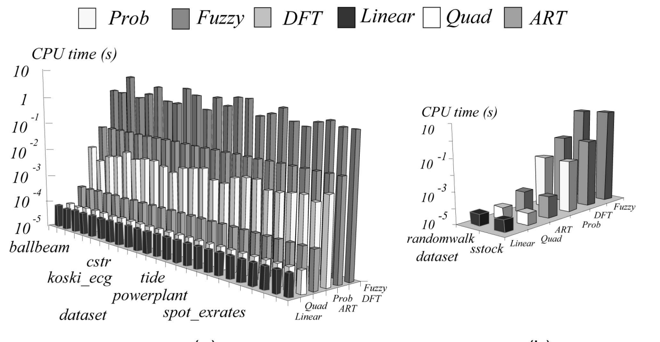


Fig. 14. Prediction efficiency versus data sets. (a) Twenty-four *benchmark* data sets (CPU time). (b) *Sstock* and *randomwalk* (CPU time).

four methods have higher errors when Δt is large. The *linear*, *quadratic*, and *DFT* approaches show nice ability of predicting short-term values. However, their errors dramatically increase with Δt, whereas the *probabilistic* one does not increase so fast. In order to further validate the effectiveness of our methods, we run the same set of experiments on the 24 *benchmark*, *sstock*, and *randomwalk* data sets. In particular, 24 *benchmark* consists of 24 real data sets, each containing 200 time series of length 256, where we set $\Delta t = 256$ and $H = 512$. The *sstock* data set is real stock data and contains 193 company stocks' daily closing price from late 1993 to early 1996, each consisting of 512 values. The *randomwalk* data set is synthetic, containing time series of length 512. For both *sstock* and *randomwalk*, we let $\Delta t = 512$ and $H = 1,024$. Furthermore, we also compare our proposed approaches with the *fuzzy* method [25] and the *ART* method [31]. Recall that for the *ART* method, we construct a decision tree over training data sets (subsequences from series) by using the *WinMine Toolkit* software [7] and compute an AR model for the data in each leaf node of the decision tree in order to predict future values. The experimental results are depicted in Fig. 13, which shows better performance of the *probabilistic* solution than that of *linear*, *quadratic*, *DFT*, *fuzzy*, and *ART*. Note that for the reason of clear illustrations, we sort the 24 *benchmark* data sets in the chart by their errors of *linear prediction*. The *prediction error* of the *DFT* method is not very stable. That is, sometimes, it achieves the second lowest error only after *probabilistic*, but sometimes, it achieves the worst of all methods. This is because different data sets have their own characteristics. If the underlying data properties change dramatically from the historical to the future value in the *frequency domain*, it is very likely that the *DFT*-based prediction would give inaccurate results.

Fig. 14 illustrates the CPU time of these prediction approaches on the 24 *benchmark*, *sstock*, and *randomwalk* data sets. Note that for the *probabilistic* approach, we also include the average construction time of indexes in CPU time. In general, the CPU times of *linear* and *quadratic* are the lowest, whereas that of *probabilistic* is higher than that of *ART* (excluding the time for constructing the *ART*) but lower than those of *DFT* and *fuzzy* by orders of magnitude. Note that although *ART* requires less time to predict future values than the *probabilistic* approach, it does not include the time for constructing the *ART* on training data, which is costly (for about several seconds, much larger compared to 10 *ms* for the *probabilistic* approach), inefficient for stream processing, and not adaptive to the change of stream data.

6.3 Query Accuracy

Next, we demonstrate the *query accuracy* of the similarity search in future time series. Specifically, we run our experiments on two data sets, *sstock* and *randomwalk*. We divide each data set into 128 time series, that is, $n = 128$. Consistent to previous settings, we choose $H = 1,024$ and $\Delta t = 512$ and select $L = 384$ as the query length $(1,024 \geq 512 + 384)$. Therefore, during each blocked period, there are a total of $n \cdot \Delta t (= 128 \times 512)$ predicted values (*future subsequences* of length 384). We randomly extract 128 subsequences of length 384 from each data set and use them as our query sequences.

Fig. 15 illustrates the impact of the similarity threshold ε on the *query accuracy* in terms of the *recall ratio* defined in (3). Here, the *query accuracy* mainly depends on the *prediction accuracy*. Furthermore, since people are usually interested in finding future series that follow certain patterns with false dismissals as few as possible, our major concern is a high *recall ratio*. Interestingly, for both data sets, the recall ratio of the *probabilistic* approach is much better than those of *linear*, *quadratic*, and *DFT* predictions. When the value of ε increases, the query result contains more candidates. Therefore, the *recall ratio* appears to be increasing for all the four approaches (the last half of the curve in the *probabilistic* one). Note that due to the data characteristics, the *recall ratio* of *randomwalk* with *DFT* remains nearly the same. When ε is very small, however, both *linear* and *quadratic* have very few candidates, that is, close to zero candidate on the average, thus leading to a near-zero recall ratio. In contrast, the results of the *DFT* and *probabilistic*

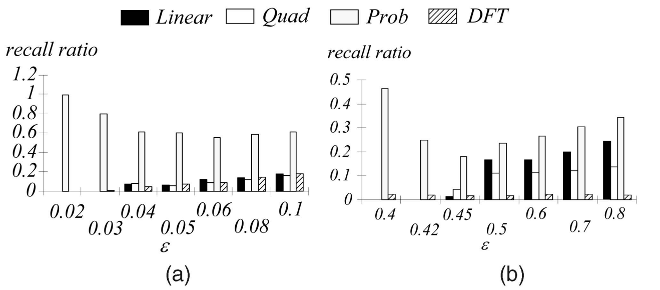


Fig. 15. Query accuracy versus ε. (a) *Sstock*. (b) *Randomwalk*.

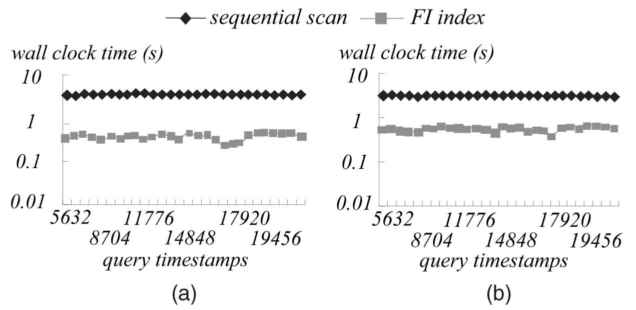


Fig. 16. Query efficiency versus time stamps. (a) *Sstock*. (b) *Randomwalk*.

approaches contain more candidates with a small ε, thus having a higher *recall ratio*.

6.4 Query Efficiency

In this set of experiments, we evaluate the *query efficiency* of the similarity search in terms of the *wall clock time* of each query. Specifically, we compare the similarity search on the B^{+}-tree index **FI** with that of the *sequential scan*. In particular, the *wall clock time* consists of two parts, CPU time and I/O cost, where each page access (I/O) is penalized by 10 *ms* [38], [39]. For the sake of fair comparisons, for the similarity search over index **FI**, we also include the index construction time of **FI** into the *wall clock time* of each query. We set the page size to 1,024 bytes and run the experiments on data sets *sstock* and *randomwalk*, respectively, where $n = 128$, $H = 1,024$, $\Delta t = 512$, and $L = 384$. Similar to the previous experiment settings, we extract 128 query sequences of length 384. The value of ε is chosen so that the selectivity of the query result is about 0.1 percent. Fig. 16 illustrates the *wall clock time* at different query time stamps, where the vertical axis is in log scale. From the experimental results, we can see that our approach outperforms the *sequential scan* by an order of magnitude in terms of the *wall clock time*.

6.5 Scalability

In this section, we evaluate the scalability of our proposed *probability* approach with respect to the number n of stream time series in terms of the average *wall clock time* of queries. Specifically, Fig. 17 illustrates the experimental results over data sets *sstock* and *randomwalk*, where $n = 64, 96, 128, 192, 256$, $H = 1,024$, $\Delta t = 512$, and $L = 384$. In the figure, we can see that our approach is always better than *sequential scan* by an order of magnitude in terms of the *wall clock time* with different numbers of time series, which confirms the scalability of our *probabilistic* method. In summary, the *probabilistic* approach works well on the *prediction accuracy* and also achieves good *query accuracy* and *efficiency*.

6.6 Performance of the Group Probabilistic Approach

Finally, we evaluate the performance of the *group probabilistic* approach over correlated time series. Specifically, we first obtain a time series S of length 10,240 by concatenating sequences of length 512 in the *sstock* (*randomwalk*) data set and then generate 127 other correlated time series by adding noise to value $S[t]$ at each time stamp t, following a

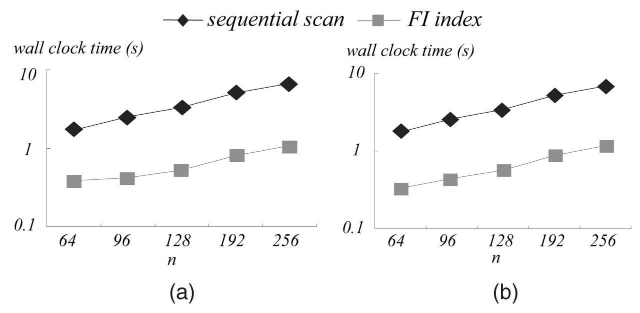


Fig. 17. The scalability test of query processing. (a) *Sstock*. (b) *Randomwalk*.

Gaussian distribution with mean zero and variance σ. Fig. 18 illustrates the *prediction error* and *recall ratio* (numbers over columns in figures) of the *group probabilistic approach* compared with the *probabilistic* approach, where $n = 128$, $H = 1,024$, $\Delta t = 512$, $L = 384$, and the query selectivity is set to 0.1 percent. From the figure, we find that when the variance σ increases (that is, series become less correlated), the *prediction error* of the *group probabilistic* approach increases, whereas that of the *probabilistic* approach is insensitive to σ. Moreover, the query accuracy of the *group probabilistic* approach in terms of the *recall ratio* is higher than that of the *probabilistic* approach due to the prediction correction with the correlation information.

7 Conclusions

In this paper, we address the issues of similarity search over future stream time series. In many applications, data often arrive periodically or in batches. In order to offer a reasonably accurate answer, we need methods to predict values and carry out similarity searches over these predicted values. We propose three prediction techniques, *polynomial*, *DFT*, and (*group*) *probabilistic*, to predict future values in the stream environment. We also present efficient indexes to facilitate the (*group*) *probabilistic* approach with both prediction tasks and similarity searches in future time series.

Acknowledgments

This work was supported in part by the Hong Kong Research Grants Council (RGC) Grant 611907 and the National Grand Fundamental Research 973 Program of China Grant 2006CB303000.

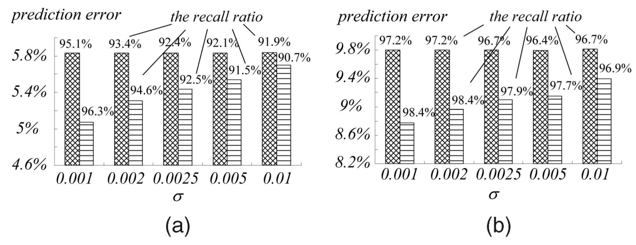


Fig. 18. Performance of the *group probabilistic* approach. (a) *Sstock*. (b) *Randomwalk*.

REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient Similarity Search in Sequence Databases," *Proc. Fourth Int'l Conf. Foundations of Data Organization and Algorithms (FODO '93),* 1993.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems," *Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems (PODS '02),* 2002.

[3] D.J. Berndt and J. Clifford, "Finding Patterns in Time Series: A Dynamic Programming Approach," *Advances in Knowledge Discovery and Data Mining,* 1996.

[4] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R^*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. ACM SIGMOD,* 1990.

[5] J.S. Boreczky and L.A. Rowe, "Comparison of Video Shot Boundary Detection Techniques," *Proc. Int'l Symp. Storage and Retrieval for Image and Video Databases,* 1996.

[6] A. Bulut and A.K. Singh, "A Unified Framework for Monitoring Data Streams in Real Time," *Proc. 21st Int'l Conf. Data Eng. (ICDE '05),* 2005.

[7] D.M. Chickering, "The WinMine Toolkit," Technical Report MSR-TR-2002-103, Microsoft, 2003.

[8] C. Cranor, T. Johnson, and O. Spatscheck, "Gigascope: A Stream Database for Network Applications," *Proc. ACM SIGMOD,* 2003.

[9] L. Chen and R. Ng, "On the Marriage of Lp-Norms and Edit Distance," *Proc. 30th Int'l Conf. Very Large Data Bases (VLDB '04),* 2004.

[10] Y. Cai and R. Ng, "Indexing Spatio-Temporal Trajectories with Chebyshev Polynomials," *Proc. ACM SIGMOD,* 2004.

[11] L. Chen, M.T. Ozsu, and V. Oria, "Robust and Fast Similarity Search for Moving Object Trajectories," *Proc. ACM SIGMOD,* 2005.

[12] G. Das, D. Gunopulos, and H. Mannila, "Finding Similar Time Series," *Proc. First European Symp. Principles of Data Mining and Knowledge Discovery (PKDD '97),* 1997.

[13] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast Subsequence Matching in Time-Series Databases," *Proc. ACM SIGMOD,* 1994.

[14] L. Gao, Z. Yao, and X. Wang, "Evaluating Continuous Nearest Neighbor Queries for Streaming Time Series via Pre-Fetching," *Proc. 11th Int'l Conf. Information and Knowledge Management (CIKM '02),* 2002.

[15] L. Gao and X. Wang, "Continually Evaluating Similarity-Based Pattern Queries on a Streaming Time Series," *Proc. ACM SIGMOD,* 2002.

[16] I. Gath and A.B. Geva, "Unsupervised Optimal Fuzzy Clustering," *IEEE Trans. Pattern Analysis and Machine Intelligence,* 1989.

[17] L. Gyorfi, G. Lugosi, and G. Morvai, "A Simple Randomized Algorithm for Sequential Prediction of Ergodic Time Series," *IEEE Trans. Information Theory,* 1999.

[18] T.V. Gestel, J. Suykens, D.E. Baestaens, A. Lambrechts, G. Lanckriet, B. Vandaele, D.B. Moor, and J. Vandewalle, "Financial Time Series Prediction Using Least Squares Support Vector Machines within the Evidence Framework," *IEEE Trans. Neural Networks,* 2001.

[19] Y.W. Huang and P.S. Yu, "Adaptive Query Processing for Time-Series Data," *Proc. Fifth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '99),* 1999.

[20] K. Kalpakis, D. Gada, and V. Puttagunta, "Distance Measures for Effective Clustering of ARIMA Time-Series," *Proc. Int'l Conf. Data Mining (ICDM '01),* 2001.

[21] K. Kanth, D. Agrawal, and A. Singh, "Dimensionality Reduction for Similarity Searching in Dynamic Databases," *Proc. ACM SIGMOD,* 1998.

[22] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani, "Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases," *Proc. ACM SIGMOD,* 2001.

[23] E. Keogh, "Exact Indexing of Dynamic Time Warping," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02),* 2002.

[24] E. Keogh, S. Lonardi, and W. Chiu, "Finding Surprising Patterns in a Time Series Database in Linear Time and Space," *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '02),* 2002.

[25] I. Kim and S.R. Lee, "A Fuzzy Time Series Prediction Method Based on Consecutive Values," *Proc. IEEE Int'l Fuzzy Systems Conf.,* 1999.

[26] S. Kim, S. Park, and W. Chu, "An Indexed-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases," *Proc. 17th Int'l Conf. Data Eng. (ICDE '01),* 2001.

[27] M. Kontaki and A.N. Papadopoulos, "Efficient Similarity Search in Streaming Time Series," *Proc. 16th Int'l Conf. Scientific and Statistical Database Management (SSDBM '04),* 2004.

[28] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A Symbolic Representation of Time Series, with Implications for Streaming Algorithms," *Proc. Eighth ACM SIGMOD Workshop Research Issues in Data Mining and Knowledge Discovery (DMKD '03),* 2003.

[29] S. Lin, M.T. Ozsu, V. Oria, and R. Ng, "An Extendible Hash for Multi-Precision Similarity Querying of Image Databases," *Proc. 27th Int'l Conf. Very Large Data Bases (VLDB '01),* 2001.

[30] D.J.C. MacKay, "Bayesian Interpolation," *Neural Computation,* 1992.

[31] C. Meek, D.M. Chickering, and D. Heckerman, "Autoregressive Tree Models for Time-Series Analysis," *Proc. Second SIAM Int'l Conf. Data Mining (SDM '02),* 2002.

[32] S. Policker and A. Geva, "A New Algorithm for Time Series Prediction by Temporal Fuzzy Clustering," *Proc. 15th Int'l Conf. Pattern Recognition (ICPR '00),* 2000.

[33] I. Popivanov and R. Miller, "Similarity Search over Time Series Data Using Wavelets," *Proc. 18th Int'l Conf. Data Eng. (ICDE '02),* 2002.

[34] Y.T. Qian, S. Jia, and W. Si, "Markov Model Based Time Series Similarity Measuring," *Proc. Int'l Conf. Machine Learning and Cybernetics,* 2003.

[35] Y. Qu, C. Wang, L. Gao, and X.S. Wang, "Supporting Movement Pattern Queries in User-Specified Scales," *IEEE Trans. Knowledge and Data Eng.,* 2003.

[36] K. Rose, E. Gurewitz, and G. Fox, "A Deterministic Annealing Approach to Clustering," *IEEE Pattern Recognition Letters,* 1990.

[37] T. Seidl and H. Kriegel, "Optimal Multi-Step k-Nearest Neighbor Search," *Proc. ACM SIGMOD,* 1998.

[38] Y. Tao, D. Papadias, and X. Lian, "Reverse kNN Search in Arbitrary Dimensionality," *Proc. 30th Int'l Conf. Very Large Data Bases (VLDB '04),* 2004.

[39] Y. Tao, D. Papadias, X. Lian, and X. Xiao, "Multidimensional Reverse kNN Search," *VLDB J.,* 2005.

[40] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering Similar Multidimensional Trajectories," *Proc. 18th Int'l Conf. Data Eng. (ICDE '02),* 2002.

[41] R. Vilalta and S. Ma, "Predicting Rare Events in Temporal Domains," *Proc. Int'l Conf. Data Mining (ICDM '02),* 2002.

[42] C.Z. Wang and X. Wang, "Supporting Content-Based Searches on Time Series via Approximation," *Proc. 12th Int'l Conf. Scientific and Statistical Database Management (SSDBM '00),* 2000.

[43] L. Wang, K.K. Teo, and Z. Lin, "Predicting Time Series with Wavelet Packet Neural Networks," *Proc. Int'l Joint Conf. Neural Network (IJCNN '01),* 2001.

[44] H. Wu, B. Salzberg, and D. Zhang, "Online Event-Driven Subsequence Matching over Financial Data Streams," *Proc. ACM SIGMOD,* 2004.

[45] W. Xue, Q. Luo, L. Chen, and Y. Liu, "Contour Map Matching for Event Detection in Sensor Networks," *Proc. ACM SIGMOD,* 2006.

[46] B. Yi and C. Faloutsos, "Fast Time Sequence Indexing for Arbitrary Lp Norms," *Proc. 26th Int'l Conf. Very Large Data Bases (VLDB '00),* 2000.

[47] B-K. Yi, H. Jagadish, and C. Faloutsos, "Efficient Retrieval of Similar Time Sequences under Time Warping," *Proc. 14th Int'l Conf. Data Eng. (ICDE '98),* 1998.

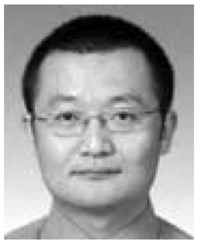
[48] Q. Zhang and A. Benveniste, "Wavelet Networks," *IEEE Trans. Neural Networks,* 1992.

[49] Y. Zhu and D. Shasha, "StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02),* 2002.

[50] Y. Zhu and D. Shasha, "Warping Indexes with Envelope Transforms for Query by Humming," *Proc. ACM SIGMOD,* 2003.

[51] Y. Zhu and D. Shasha, "Efficient Elastic Burst Detection in Data Streams," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '03),* 2003.

Xiang Lian received the BS degree from the Department of Computer Science and Technology, Nanjing University, in 2003. He is currently a PhD candidate in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong. His research interests include stream time series and probabilistic databases. He is a student member of the IEEE.



Lei Chen received the BS degree in computer science and engineering from Tianjin University, China, in 1994, the MA degree from the Asian Institute of Technology, Thailand, in 1997, and the PhD degree in computer science from the University of Waterloo, Canada, in 2005. He is now an assistant professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include multimedia and time series databases, sensor and peer-to-peer databases, and stream and probabilistic databases. He is a member of the IEEE.