A Data-adaptive and Dynamic Segmentation Index for Whole Matching on Time Series [*]

Yang Wang[†] Peng Wang[†] Jian Pei[‡] Wei Wang[†] Sheng Huang[§]

[†] School of Computer Science, Fudan University, Shanghai, China
[‡] School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
[§] Information Management Team, IBM Research China, Shanghai, China

[†] {081024004, pengwang5, weiwang1}@fudan.edu.cn
[‡] jpei@cs.sfu.ca [§] huangssh@cn.ibm.com

ABSTRACT

Similarity search on time series is an essential operation in many applications. In the state-of-the-art methods, such as the R-tree based methods, SAX and iSAX, time series are by default divided into equi-length segments globally, that is, all time series are segmented in the same way. Those methods then focus on how to approximate or symbolize the segments and construct indexes. In this paper, we make an important observation: global segmentation of all time series may incur unnecessary cost in space and time for indexing time series. We develop DSTree, a data adaptive and dynamic segmentation index on time series. In addition to savings in space and time, our new index can provide tight upper and lower bounds on distances between time series. An extensive empirical study shows that our new index DSTree supports time series similarity search effectively and efficiently.

1. INTRODUCTION

Similarity search on time series is essential in many applications [10]. Given a set $\mathcal{TS}$ of time series, a query time series Q, and a distance threshold ϵ, a *similarity search* retrieves the time series $S \in \mathcal{TS}$ such that $D(Q, S) \leq \epsilon$, where $D(\cdot, \cdot)$ is a distance function. When the Euclidean distance is used and the time series in question are assumed of the same length, the problem is called *whole matching* [1], which has been popularly used in various applications. The problem is challenging in practice, since often the set of time series $\mathcal{TS}$ to be searched may contain many time series and each time series may be long. To tackle the whole matching problem, many index structures have been proposed [1, 4, 5, 2, 16, 3], which will be briefly reviewed in Section 6, all those indexes are based on two fundamental principles.

[*]We sincerely thank Dr. Themis Palpanas for sending us the iSAX 2.0 code. We are deeply grateful to the anonymous reviewers for their insightful and constructive comments and suggestions that help to improve the quality of this paper. We tried our best to accommodate their suggestions in this camera-ready version. The work is supported in part by NSFC under grants 61103009, 61170006, 61033010, IBM-Fudan Joint Study program JSA201212006, an NSERC Discovery Grant and a BCFRST NRAS Endowment Research Team Program project. All opinions, findings, conclusions and recommendations in this paper are those of the author and do not necessarily reflect the views of the funding agencies.

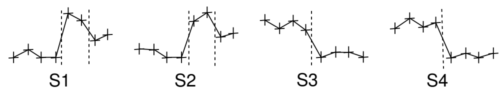


Figure 1: Dynamic segmentation of time series.

Principle 1: Dimensionality Reduction by Global Segmentation

A time series can be regarded as a point in a multidimensional space, one dimension representing a time instant. A fundamental challenge, however, is that the length of time series is often long. A time series often contains readings at hundreds or even thousands of instants. It is highly ineffective to directly index time series using spatial indexes, such as an R-tree [7].

To tackle this problem, many existing methods apply dimensionality reduction techniques, such as Singular Value Decomposition (SVD) [8], Discrete Fourier Transform (DFT) [1], Discrete Wavelet Transform (DWT) [4], Piecewise Linear Approximation (PLA) [14], Piecewise Aggregate Approximation (PAA) [21], Adaptive Piecewise Constant Approximation (APCA) [21] and Chebyshev Polynomials (CP) [2]. After dimensionality reduction, a multidimensional index, such as R-tree [7], can be used as an index in the lower dimensional space.

Accordingly, in the state-of-the-art time series indexing methods, such as the R-tree based methods, SAX [15], and iSAX [16], *all time series to be indexed are segmented in the same way*. Thus, they are *global segmentation approaches*. Those methods focus on how to approximate or symbolize segments and construct indexes. The segmentation of time series is not closely integrated with index building. Does such a global segmentation method provide the best benefit to time series indexing?

EXAMPLE 1 (SEGMENTATION). *Consider the 4 time series in Figure 1. Each time series has 8 time instants. To reduce the dimensionality, we can segment each time series into 4 segments, each segment consisting of 2 instants.*

If we notice that time series $S1$ and $S2$ have (relatively) stable values on the first 4 instants, we can segment $S1$ and $S2$ into 3 segments: the first segments consist of the first 4 instants, the second segments consist of the 5th and the 6th instants, and the last segments consist of the 7th and the 8th instants, as indicated by the dotted lines in the figure. To the contrast, time series $S3$ and $S4$ have (relatively) stable values both on the first 4 instants and on the last 4 ones. Accordingly, we can segment them such that the first *segments cover the first* 4 *instants and the second segments cover the last* 4 *instants.*

By dynamic segmentations adaptive to data, we can reduce dimensionality further, in this example, from 4 *to* 3 *for* $S1$ *and* $S2$*, and to* 2 *for* $S3$ *and* $S4$*,. At the same time, we can retain good approximation quality.* ■

Example 1, though simple, clearly shows that *local segmentation enables substantial opportunities for more effective indexes*. If we can segment time series in an adaptive way, we may be able to achieve better dimensionality reduction and thus save more space and query answering time.

Now, the challenge is *how we can dynamically segment time series in a data adaptive manner, and retain good quality*.

Principle 2: Using Lower Bounds in Search

Dimensionality reduction almost unavoidably comes with errors in data representation. An essential requirement in similarity search, however, is "no false dismissals". The *lower bounding property* (also known as the *contractive property*) is an important desirable property for the dimensionality reduction representation methods of time series. A dimensionality reduction method is said to hold the lower bounding property if the method comes with a distance lower bound function $D_{LB}(\tilde{S_1}, \tilde{S_2}) \leq D(S_1, S_2)$ for any time series S_1 and S_2, where $\tilde{S_1}$ and $\tilde{S_2}$ are the approximation representations of S_1 and S_2, respectively, in the method. A method with the lower bounding property guarantees no false negative in search. That is, when a time series S is pruned using the lower bound function $D_{LB}(\tilde{Q}, \tilde{S}) > \epsilon$, since $D(Q, S) \geq D_{LB}(\tilde{S_1}, \tilde{S_2}) > \epsilon$, S is definitely not an answer to the similarity search.

While lower bounding is well explored in literature, to the best of our knowledge, *no existing methods consider using upper bounds in similarity search systematically*. If a method comes with a distance upper bound function $D_{UB}(\tilde{S_1}, \tilde{S_2}) \geq D(S_1, S_2)$ for any time series S_1 and S_2, where $\tilde{S_1}$ and $\tilde{S_2}$ are the approximation representations of S_1 and S_2, respectively, in the method, then once $D_{UB}(\tilde{Q}, \tilde{S}) < \epsilon$, we can immediately know S is an answer to the similarity search without computing the exact distance $D(Q, S)$. Although some previous methods propose upper bound for time series similarity computation [17], they only consider how to define and compute the upper bound of the distance between two time series. How to utilize the upper bound in the index for a large number of time series is far from trivial and has not been solved.

Moreover, upper bounding can be used to answer interesting queries beyond similarity search. For example, consider query "what is the distribution of the distance between Q and the time series in the database?" With both lower bounding and upper bounding, we may be able to give a bounded histogram quickly as the answer to the question based on the index only without accessing the original data. An application example of the histogram obtained as such is to help to set a meaningful threshold in similarity search.

Now, the challenge is *how to develop an effective upper bounding mechanism in indexes for efficient similarity search*.

In this paper, we study the *whole matching* problem [1] where Euclidean distance is used and time series have the same length. It is a fundamental time series processing problem tackled by numerous previous studies [1, 4, 5, 2, 16, 3]. Please note that our work can be easily extended to *subsequence matching* [6] where query time series are allowed to have different lengths.

We explore data-adaptive dynamic segmentation and upper bounding in time series indexes. We propose a new representation of time series that is an extension of the renowned Adaptive Piecewise Constant Approximation (APCA). It not only offers better representation accuracy, but also supports upper bound estimation, which enriches the functionality of index greatly.

Symbol	Meaning
S, S'	time series
$\|S\|$	the length of time series S
$D(S, S')$	the (Euclidean) distance between time series S and S'
$D_{LB}(N, Q)$, $D_{UB}(N, Q)$	the lower and upper bound of distance between time series Q and time series in node N
$S = (S_1, \ldots, S_m)$	a time series is divided into m segments
r_j	the right end time instant of segment j
μ_j^S	the mean of segment j in time series S
σ_j^S	the standard deviation of segment j in time series S
SG	a segmentation of a time series
C	the number of time series indexed in the subtree rooted at a node
Z	the synopsis at a node
ψ	Leaf capacity of DSTree
N	a node in a DSTree
$\mathcal{TS}_N$	the time series assigned to node N
LB_i^μ, LB_i^σ, UB_i^μ, UB_i^σ	the lower and upper bounds using mean or standard deviation, see Equations 11, 12, 13, and 14 for detail
H-split$^{\text{M}}$	H-split using mean value
H-split$^{\text{SD}}$	H-split using standard deviation
V-split$^{\text{L}}$	V-split using the left subsegment
V-split$^{\text{R}}$	V-split using the right subsegment

Table 1: Some frequently used symbols.

We develop DSTree, a data adaptive and dynamic segmentation index on time series. In addition to savings in space and time, our new index can provide tight upper and lower bounds on distances between time series. An extensive empirical study shows that our new index DSTree supports time series similarity search effectively and efficiently.

The rest of the paper is organized as follows. Section 2 introduces the new representation of time series. Section 3 develops the new index and the construction method. Section 4 applies the new index in similarity search. Section 5 reports the experiment results. Section 6 reviews the related work. Section 7 concludes the paper. Table 1 summarizes the symbols frequently used in this paper.

2. EXTENDING APCA REPRESENTATION

In this section, we extend the well-known *Adaptive Piecewise Constant Approximation* (APCA) of time series data. Our extension, called EAPCA, will be used to represent time series in our index. We also derive upper and lower bounds of distances among time series using the extended approximation.

2.1 APCA

A *time series* $S = (s_1, \ldots, s_n)$ is a sequence of values. Without loss of generality, in this paper we assume that every time series has a value at every time instant $t = 1, 2, \ldots, n$. We denote by $|S| = n$ the *length* of the time series S, and by $S[i] = s_i$ $(1 \leq i \leq |S|)$ the value of S at time instant $t = i$.

Given two time series S and S' such that $|S| = |S'|$, the *(Euclidean) distance* between S and S' is $D(S, S') = \sqrt{\sum_{i=1}^{|S|}(S[i] - S'[i])^2}$. The Euclidean distance is popularly used in time series analysis. Moreover, there are strong evidences showing that the Euclidean distance is superior in accuracy comparing to other similarity measures [13, 18, 20]. In the rest of the paper, we assume the Euclidean distance, and, when the distance between two time series is concerned, the time series have the same length.

In many applications, it is highly desirable to estimate the distance between two time series quickly. There are existing methods providing lower bounds by segmenting time series. Here, we review a popularly used method, APCA.

APCA divides a time series $S = (s_1, \ldots, s_n)$ into several disjoint segments, $S = (S_1, \ldots, S_m)$, $(m \leq n)$, where $S_j = (s_{r_{j-1}+1}, \ldots, s_{r_j})$ $(1 \leq j \leq m, r_0 = 0, 1 \leq r_1 < \cdots < r_m = n)$. APCA approximates each segment S_j by a pair (μ_j, r_j), where $\mu_j = \frac{\sum_{k=r_{j-1}+1}^{r_j} s_k}{r_j - r_{j-1}}$ is the mean value of the segment. That is, S can be approximated as $\tilde{S} = ((\mu_1, r_1), \ldots, (\mu_m, r_m))$.

For two time series X and Y such that $|X| = |Y|$, let $\tilde{X} = ((\mu_1^X, r_1), \ldots, (\mu_m^X, r_m))$ and $\tilde{Y} = ((\mu_1^Y, r_1), \ldots, (\mu_m^Y, r_m))$ be the APCA representations of X and Y, respectively. The segmentations of the two time series are said to be *aligned* in $\tilde{X}$ and $\tilde{Y}$, since$\tilde{X}$ and $\tilde{Y}$ use the same $r_1, \ldots, r_m$ and $r_0 = 0$.

Using the mean values, APCA can give a lower bound of the distance between two time series. Apparently, we have the following.

LEMMA 1 (APCA LOWER BOUND). *Given two time series X and Y such that $|X| = |Y|$, let $\tilde{X} = ((\mu_1^X, r_1), \ldots, (\mu_m^X, r_m))$ and $\tilde{Y} = ((\mu_1^Y, r_1), \ldots, (\mu_m^Y, r_m))$ be two aligned APCA representations of X and Y, respectively. Then,*

$$D(X, Y) \geq \sqrt{\sum_{i=1}^{m} (r_i - r_{i-1})(\mu_i^X - \mu_i^Y)^2} \tag{1}$$

■

Equipped with only mean values, APCA cannot provide any upper bound on distance between time series. Next, we show that combining standard deviations we can derive an upper bound and a tighter lower bound on distances.

2.2 EAPCA and Upper/Lower Bounds Using Standard Deviations

We can extend APCA by including the standard deviation for every segment. Concretely, for a time series $S = (s_1, \ldots, s_n)$ and an APCA representation $\tilde{S} = ((\mu_1, r_1), \ldots, (\mu_m, r_m))$, we extend the approximation to the *extended APCA representation* (*EAPCA* for short), denote by $\tilde{S} = ((\mu_1, \sigma_1, r_1), \ldots, (\mu_m, \sigma_m, r_m))$, where $\sigma_i = \sqrt{\frac{\sum_{j=r_{i-1}+1}^{r_i} s_j^2}{r_i - r_{i-1}} - (\frac{\sum_{j=r_{i-1}+1}^{r_i} s_j}{r_i - r_{i-1}})^2}$ is the standard deviation of the i-th segment $(1 \leq i \leq m)$.

We have the following results.

THEOREM 1 (BOUNDS). *Given two time series X and Y such that $|X| = |Y|$, let $\tilde{X} = ((\mu_1^X, \sigma_1^X, r_1), \ldots, (\mu_m^X, \sigma_m^X, r_m))$ and $\tilde{Y} = ((\mu_1^Y, \sigma_1^Y, r_1), \ldots, (\mu_m^Y, \sigma_m^Y, r_m))$ be two aligned EAPCA representations of X and Y, respectively. Then,*

$$D(X, Y) \geq \sqrt{\sum_{i=1}^{m} (r_i - r_{i-1})[(\mu_i^X - \mu_i^Y)^2 + (\sigma_i^X - \sigma_i^Y)^2]} \tag{2}$$

and

$$D(X, Y) \leq \sqrt{\sum_{i=1}^{m} (r_i - r_{i-1})[(\mu_i^X - \mu_i^Y)^2 + (\sigma_i^X + \sigma_i^Y)^2]} \tag{3}$$

The lower and upper bounds in Equations 2 and 3 are realizable.

PROOF.

$$\begin{aligned} D(X, Y)^2 &= \sum_{i=1}^{m} \sum_{j=r_{i-1}+1}^{r_i} (x_j - y_j)^2 \\ &= \sum_{i=1}^{m} (r_i - r_{i-1}) \mu_i^{(X-Y)^2} \\ &= \sum_{i=1}^{m} (r_i - r_{i-1})[(\mu_i^X - \mu_i^Y)^2 + (\sigma_i^{X-Y})^2] \end{aligned} \tag{4}$$

where $\mu_i^{(X-Y)^2} = \frac{\sum_{j=r_{i-1}+1}^{r_i} (x_j - y_j)^2}{r_i - r_{i-1}}$ and $\sigma_i^{X-Y} = \sqrt{\frac{\sum_{j=r_{i-1}+1}^{r_i} (x_j - y_j)^2}{r_i - r_{i-1}} - (\frac{\sum_{j=r_{i-1}+1}^{r_i} (x_j - y_j)}{r_i - r_{i-1}})^2}$. Due to the definition of standard deviation, we have

$$\begin{aligned} (\sigma_i^{X-Y})^2 &= (\sigma_i^X)^2 + (\sigma_i^Y)^2 - 2Cov(X_i, Y_i) \\ &= (\sigma_i^X)^2 + (\sigma_i^Y)^2 - 2\rho(X_i, Y_i)\sigma_i^X \sigma_i^Y \end{aligned} \tag{5}$$

where

$$Cov(X_i, Y_i) = \frac{\sum_{j=r_{i-1}+1}^{r_i} (x_j - \mu_i^X)(y_j - \mu_i^Y)}{r_i - r_{i-1}} \tag{6}$$

is the covariance between X_i and Y_i, and

$$\rho(X_i, Y_i) = \frac{\sum_{j=r_{i-1}+1}^{r_i} (x_j - \mu_i^X)(y_j - \mu_i^Y)}{\sqrt{\sum_{j=r_{i-1}+1}^{r_i} (x_j - \mu_i^X)} \sqrt{\sum_{j=r_{i-1}+1}^{r_i} (y_j - \mu_i^Y)}} \tag{7}$$

is the correlation coefficient between segments X_i and Y_i. Since $-1 \leq \rho(X_i, Y_i) \leq 1$, we have

$$(\sigma_i^X - \sigma_i^Y)^2 \leq (\sigma_i^{X-Y})^2 \leq (\sigma_i^X + \sigma_i^Y)^2 \tag{8}$$

Combining Equations 8 and 4, we have Equations 2 and 3. ■

Comparing Equations 1 and 2, the lower bound given by EAPCA uses the standard deviations to achieve a tighter bound. The bounds are realizable.

2.3 Bounding Distances to a Set of Time Series

Often, we need to estimate the distance between a time series and a set of time series. We can infer the lower and upper bounds of the distance based on Equations 2 and 3.

For a time series X and a set of time series $Y_1, \ldots, Y_l$ $(|X| = |Y_1| = \cdots = |Y_l|)$, let $\tilde{X} = ((\mu_1^X, \sigma_1^X, r_1), \ldots, (\mu_m^X, \sigma_m^X, r_m))$, $\tilde{Y_1} = ((\mu_1^{Y_1}, \sigma_1^{Y_1}, r_1), \ldots, (\mu_m^{Y_1}, \sigma_m^{Y_1}, r_m))$, $\ldots$, $\tilde{Y_l} = ((\mu_1^{Y_l}, \sigma_1^{Y_l}, r_1), \ldots, (\mu_m^{Y_l}, \sigma_m^{Y_l}, r_m))$ be aligned EACPA representations, respectively. Let the minimal and maximal mean values in the i-th segments of $Y_1, \ldots, Y_l$, respectively, be $\mu_i^{min} = \min_{1 \leq j \leq l}\{\mu_i^{Y_j}\}$ and $\mu_i^{max} = \max_{1 \leq j \leq l}\{\mu_i^{Y_j}\}$. Moreover, let the minimal and maximal standard deviation values in the i-th segments of $Y_1, \ldots, Y_l$, respectively, be $\sigma_i^{min} = \min_{1 \leq j \leq l}\{\sigma_i^{Y_j}\}$ and $\sigma_i^{max} = \max_{1 \leq j \leq l}\{\sigma_i^{Y_j}\}$. We have the following result.

THEOREM 2 (BOUNDS ON SET). *For aligned EAPCA representations $\tilde{X}$ of a time series X and $\tilde{Y_1}, \ldots, \tilde{Y_l}$ of a set of time series $Y_1, \ldots, Y_l$,*

$$\min_{1 \leq j \leq l}\{D(X, Y_j)\} \geq \sqrt{\sum_{i=1}^{m} (r_i - r_{i-1})(LB_i^{\mu} + LB_i^{\sigma})} \tag{9}$$

and

$$\max_{1 \le j \le l} \{D(X, Y_j)\} \le \sqrt{\sum_{i=1}^{m} (r_i - r_{i-1})(UB_i^{\mu} + UB_i^{\sigma})} \quad (10)$$

where

$$LB_i^{\mu} = \begin{cases} (\mu_i^{min} - \mu_i^X)^2 & \text{if } \mu_i^X \le \mu_i^{min}; \\ 0 & \text{if } \mu_i^{min} < \mu_i^X \le \mu_i^{max}. \\ (\mu_i^{max} - \mu_i^X)^2 & \text{if } \mu_i^{max} < \mu_i^X; \end{cases} \quad (11)$$

$$LB_i^{\sigma} = \begin{cases} (\sigma_i^{min} - \sigma_i^X)^2 & \text{if } \sigma_i^X \le \sigma_i^{min}; \\ 0 & \text{if } \sigma_i^{min} < \sigma_i^X \le \sigma_i^{max}. \\ (\sigma_i^{max} - \sigma_i^X)^2 & \text{if } \sigma_i^{max} < \sigma_i^X; \end{cases} \quad (12)$$

and,

$$UB_i^{\mu} = \begin{cases} (\mu_i^{max} - \mu_i^X)^2 & \text{if } \mu_i^X \le \frac{\mu_i^{min} + \mu_i^{max}}{2}; \\ (\mu_i^{min} - \mu_i^X)^2 & \text{if } \frac{\mu_i^{min} + \mu_i^{max}}{2} < \mu_i^X; \end{cases} \quad (13)$$

$$UB_i^{\sigma} = (\sigma_i^{max} + \sigma_i^X)^2 \quad (14)$$

PROOF. (Lower boud) From Equation 2, it is easy to see that for the i-th segment, the component $(\mu_i^X - \mu_i^Y)^2 + (\sigma_i^X - \sigma_i^Y)^2$ can be decomposed into two items: $(\mu_i^X - \mu_i^Y)^2$, which is only related to the mean value, and $(\sigma_i^X - \sigma_i^Y)^2$, which is related to the standard deviation. Since both of them are non-negative, We can obtain a lower bound from the lower bounds of these two items.

We compare μ_i^X and the range of mean values of Y_j's, $[\mu_i^{min}, \mu_i^{max}]$, and have 3 cases as follows. Case 1: If μ_i^X is smaller than the minimal mean value μ_i^{min} of Y_j's, it is obvious that for any Y_j, $(\mu_i^X - \mu_i^{Y_j})^2 \ge (\mu_i^X - \mu_i^{min})^2$. Thus, $(\mu_i^X - \mu_i^{min})^2 \le (\mu_i^X - \mu_i^Y)^2$. We denote by $LB_i^{\mu} = (\mu_i^X - \mu_i^{min})^2$. Case 2: If μ_i^X falls within the range, $(\mu_i^X - \mu_i^Y)^2 = 0$. We set $LB_i^{\mu} = 0$. Case 3: If $\mu_i^X > \mu_i^{max}$, then for any Y_j, $(\mu_i^X - \mu_i^{Y_j})^2 \ge (\mu_i^X - \mu_i^{max})^2$. We set $LB_i^{\mu} = (\mu_i^X - \mu_i^{max})^2$.

We can derive a lower bound LB_i^{σ} similarly. Combining these items, we can obtain the lower bound in the theorem.

The upper bound can be proved in a similar way. ■

Theorem 2 indicates that, for a set of time series $\{Y_1, \ldots, Y_l\}$, to compute the lower and upper bounds between the set and any other time series, we need to maintain only the minimum and maximum mean values and standard deviation of each segment for all Y_j's: μ_i^{min}, μ_i^{max}, σ_i^{min} and σ_i^{max} $(1 \le i \le m)$.

3. THE DSTREE INDEX

In this section, we develop our new *dynamic splitting tree* index (*DSTree* for short) on time series.

3.1 DSTree

One critical feature in our new DSTree is the segmentation information. In general, for a time series $S = (s_1, \ldots, s_n)$, a *segmentation* of S divides S into m exclusive segments $S = (S_1, \ldots, S_m)$, where $S_1 = (s_1, \ldots, s_{r_1})$ and $S_i = (s_{r_{i-1}+1}, \ldots, s_{r_i})$ $(i > 1, r_m = n)$. Apparently, to record a segmentation, we only need to record m, the number of segments, and $(r_1, \ldots, r_m)$, the right-endpoints of the segments, where $1 \le r_1 < \cdots < r_m = n$.

Given a time series S, let $SG_1 = (r_1, \ldots, r_m)$ and $SG_2 = (r'_1, \ldots, r'_{m'})$ be two segmentations. SG_2 is called a *one-segment refinement* of SG_1, denoted by $SG_1 \prec^1 SG_2$, if $m' = m + 1$ and there exists a number i_0 $(1 \le i_0 < m)$ such that, for all $1 \le i \le i_0$, $r_i = r'_i$; and for $i > i_0$, $r_i = r'_{i+1}$.

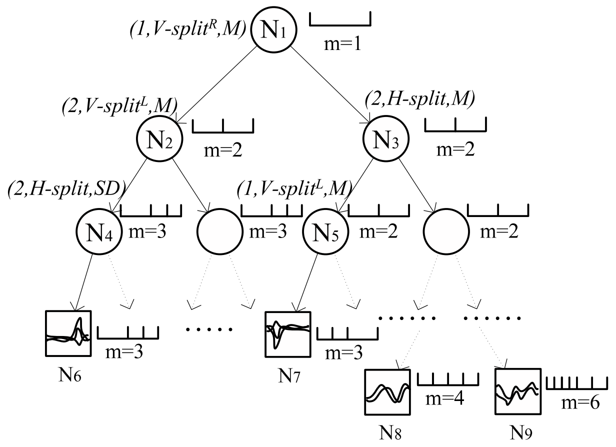


Figure 2: DSTree

EXAMPLE 2. *Consider a time series S such that $|S| = 10$, and two segmentations $SG_1 = (3, 8, 10)$ and $SG_2 = (3, 5, 8, 10)$. SG_1 divides S into 3 segments, $(1, 3), (4, 8), (9, 10)$. SG_2 divides S into 4 segments, $(1, 3), (4, 5), (6, 8), (9, 10)$. SG_2 is a one-segment refinement of SG_1 since it further divides the second segment in SG_1 into two smaller segments.* ■

We call a segmentation SG_2 a *refinement* of segmentation SG_1, denoted by $SG_1 \prec SG_2$, if there exist a series of segmentations $SG'_1, \ldots, SG'_l$ $(l \ge 2)$ such that $SG_1 = SG'_1$, $SG_2 = SG'_l$, and $SG'_i \prec^1 SG'_{i+1}$ for $(1 \le i < l)$.

As illustrated in Figure 2, a DSTree organizes time series to be indexed into a hierarchy. There are two types of nodes: *internal nodes* and *leaf nodes*. Each node contains the following information.

1. The number C of time series indexed in the subtree rooted at this node.
2. The segmentation $SG = (r_1, \ldots, r_m)$ of the time series indexed at this node, where $1 \le r_1 < \cdots < r_m = n$, and r_i $(1 \le i \le m)$ is the right-endpoint of the i-th segment.
3. A synopsis $Z = (z_1, z_2, \cdots, z_m)$, where $z_i = (\mu_i^{min}, \mu_i^{max}, \sigma_i^{min}, \sigma_i^{max})$. The synopsis is used to compute the upper and lower bounds.
4. A leaf node links to a disk file that stores up to ψ time series represented by the synopsis of this leaf node, where ψ is the *leaf capacity* of the DSTree. An internal node has two pointers pointing to children nodes.
5. An internal node stores a splitting strategy SP, which will be discussed in detail in Section 3.3.

In Figure 2, a circle represents an internal node, and a rectangle represents a leaf node, where up to $\psi = 2$ time series are stored. In the figure, the segmentation and the number of segments m are shown for each node, too.

In a DSTree, for an internal node N and its segmentation SG_N, the segmentation $SG_{N'}$ in any descendant node N' of N is either the same as SG_N or a refinement of SG_N. Consequently, different nodes in a DSTree may have different segmentations. Different segmentations may divide time series into different numbers of segments, such as the segmentations in nodes N_4 and N_5 in Figure 2. Even if two segmentations have the same number of segments, they still can be different. For example, in Figure 2, the segmentations in nodes N_4 and N_7 both have 3 segments, but the segmentations are still different.

Algorithm 1 $N.Insert(X)$: N is a node, X is a time series

```
1: update $Z$ in node $N$ according to $X$;
2: if $N$ is a leaf node then
3:     if $C < \psi$ then                          ▷ $N$ has space to hold $X$
4:         Append $X$ to data file pointed by $N$, $C = C + 1$;
5:     else                          ▷ $C == \psi$, no space in $N$ to hold $X$
6:         Append $X$ to data file pointed by $N$, $C = C + 1$;
7:         $SP = BestSplit()$;
8:         Create two children nodes for $N$;
9:         for each time series $Y$ in $N$ do
10:            $N' = N.routeToChild(Y, SP)$; $N'.insert(Y)$;
11:        end for
12:    end if
13: else
14:    $N' = N.routeToChild(X, SP)$; $N'.insert(X)$;
15: end if
```

3.2 DSTree Construction

Given a set $\mathcal{TS}$ of time series, each of length n, a DSTree is constructed in two steps as follows.

Step 1: Initialization. We initialize a DSTree that contains only the root node N_R. The segmentation $SG = (n)$, that is, each time series is regarded as containing only one segment.

Step 2: Insertion. We insert the time series in $\mathcal{TS}$ one by one into the DSTree. The insertion step is to assign every time series X to a leaf node. Ideally, similar time series are allocated to the same leaf node or a subtree, so that they can be deliberated in similarity search using the same segmentations. For the interest of computational efficiency, we heuristically follow a path from the root node to assign a time series X to a leaf node.

Specifically, for each time series $X \in \mathcal{TS}$, we first visit the root node N_R. In the case that N_R is a leaf node, we assign X to N_R if N_R has space; otherwise, we split N_R according to the splitting strategy SP of N_R, which will be discussed later in Section 3.3. If N_R is an internal node, we select the child node of N_R that X fits better, and recursively search until a leaf node is met. The pseudocode of function $Insert$ is shown in Algorithm 1.

A critical step of this algorithm is the function $BestSplit()$, which selects the best splitting strategy whenever a node is split. We provide multiple types of splitting strategies. Whenever splitting a node, we call $BestSplit()$ to find the best one, denoted as SP. Function $routeToChild()$ uses SP to determine which child node one time series belongs to. These two functions will be discussed in the next section.

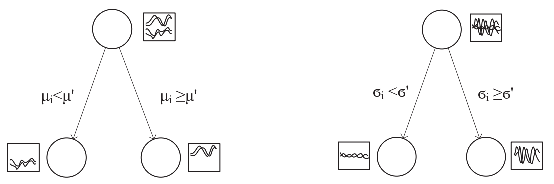
3.3 Node Splitting Strategies

At an internal node whose subtree indexes a subset of time series, there are multiple possible ways to partition the time series into smaller subsets and assign them to children nodes. We need to define a good measurement to assess the benefit of different strategies, and find a good splitting strategy. In this subsection, we first demonstrate the ideas behind various splitting strategies, and then present those strategies and a quality measure.

3.3.1 Ideas

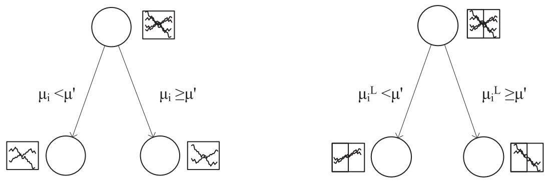
We can split a set of time series in two ways: *horizontal splitting* (*H-split* for short) and *vertical splitting* (*V-split* for short).

In an H-split, the segmentation remains unchanged, but the set of time series are split into two disjoint sets. To split, the time series are assigned to different subsets according to a selected segment. Either mean or standard deviation of the selected segment can be used to make the assignments. Two examples are shown in Figure 3, in which only the i-th segment is shown. Figure 3(b) shows an example where the time series cannot be divided well into two



(a) Splitting using mean (b) Splitting using standard deviation

Figure 3: Horizontal splitting



(a) Using H-split (b) Using V-split

Figure 4: Vertical splitting

subsets using the mean, but can be partitioned well using standard deviation.

V-split leads to a one-segment refinement of the current segmentation. We illustrate this process in Figure 4. The time series in Figure 4(a) cannot be split well using an H-split for the i-th segment, since the 4 time series have similar mean and standard deviation values. In a V-split, we first split the segment into two, and then cluster time series according to the mean of the left subsegment, as shown in Figure 4(b).

DSTree provides more possible ways to divide and conquer time series, and thus has the potential to achieve more similar time series in leaf nodes. All the state-of-the-art methods, such as the R-tree based methods and iSAX, only support horizontal splitting, and only the mean values can be used in splitting. No segmentation refinement is allowed in those methods.

3.3.2 Splitting Strategies

A selection of splitting strategies happens only when a leaf node has no space to accommodate a newly assigned time series, and thus has to be split to host two children nodes. The global user-specified parameter ψ defines the maximum number of time series that can be indexed by a leaf node. Consider a leaf node N that needs to hold a set $\mathcal{TS}_N$ of $\psi + 1$ time series, where the segmentation is SG. We need to split N into two nodes. Now we specify the splitting strategies H-split and V-split as follows.

H-split. Suppose the i-th segment is selected to be used in splitting. We will discuss the choice of segment when we discuss the quality measure in Section 3.3.3. In an H-split^{M} (for H-split using mean values), suppose the range of the mean values in the i-th segment of $\mathcal{TS}_N$ is $[\mu_i^{min}, \mu_i^{max}]$, we split N and generate two children nodes N_l and N_r with the same segmentation SG in N. The range of mean values of the i-th segment in N_l is $[\mu_i^{min}, \frac{\mu_i^{min}+\mu_i^{max}}{2})$, and that in N_r is $[\frac{\mu_i^{min}+\mu_i^{max}}{2}, \mu_i^{max}]$. The time series in N will be assigned to N_l and N_r according to their mean values.

Similarly, in an H-split^{SD} (for H-split using standard deviation values), suppose the range of the standard deviation value in the i-th segment is $[\sigma_i^{min}, \sigma_i^{max}]$, the range of standard deviation in the i-th segment in N_l is $[\sigma_i^{min}, \frac{\sigma_i^{min}+\sigma_i^{max}}{2})$, and that in N_r is

$[\frac{\sigma_i^{min}+\sigma_i^{max}}{2}, \sigma_i^{max}]$.

V-split. Suppose the i-th segment is selected to be used in splitting. Again, we will discuss the choice of segment when we discuss the quality measure in Section 3.3.3. We refine the segmentation SG by splitting the i-th segment into two equal-length segments: $S_i = [r_{i-1}+1, r_{i-1} + \lfloor \frac{r_i - r_{i-1}}{2} \rfloor]$ and $S'_i = [\lfloor r_i - \frac{r_i - r_{i-1}}{2} \rfloor + 1, r_i]$. We use one of the two new subsegments and apply an H-split to partition the time series. We denote by V-split^L and V-split^R, respectively, the left and the right subsegment is chosen. Consequently, two children nodes are created for node N. Clearly, V-split contains an H-split step.

A splitting strategy can be written as a tuple SP =(*sid*, *strategy*, *measure*), where $sid \in [1, m]$ is the segment id that is selected in the splitting, m is the number of segments in the current segmentation SG, *strategy* $\in$ {H-split, V-split^L, V-split^R} is the choice of H- or V-split (and the subsegment in the case of V-split), and *measure*$\in \{M, SD\}$ records whether the mean values or the standard deviation values are used in the H-split.

For example, in Figure 2, SP in node N_2 is (2, V-split^L, M), which means that the second segment is selected, a V-split is applied, the left subsegment and the mean values are used in the H-split. SP of node N_3 is (2, H-split, M), which means the second segment is selected, an H-split is applied using the mean values.

3.3.3 *Splitting Strategy Quality Measure*

When a node is split, the time series assigned to the node are then assigned to the two children nodes created in the splitting process. As just discussed, several different strategies can be used to make the assignment, including choosing H-split or V-split, the segment used and the measurement (mean or standard deviation). We need a quality measure to evaluate the benefit of various splitting strategies in order to choose a good one.

A brute-force method to evaluate the quality of a splitting strategy is that, for every possible strategy, we compute the similarity among the time series assigned to each child node. This brute-force method, however, is very costly. For each splitting strategy, the time complexity is $O(\psi^2)$. If there are m segments, then the total cost to find the best strategy is $O(m \times 2 \times 2 \times \psi^2) = O(m \cdot \psi^2)$.

In this subsection, we tackle the cost by using the upper and lower bounds of the time series in the children nodes to evaluate the split quality.

Given a node N in a DSTree, let Q be a query time series. The effectiveness of the upper and lower bounds in node N with respect to Q can be measured by *the bound range*, which is the difference between the upper bound and the lower bound of the distances between Q and the set of time series indexed in N, that is,

$$R(Q) = \sum_{i=1}^{m}(r_i - r_{i-1})((UB_i^\mu + UB_i^\sigma) - (LB_i^\mu + LB_i^\sigma)), \quad (15)$$

where $UB_i^\mu, UB_i^\sigma, LB_i^\mu, LB_i^\sigma$ are defined in Equations 13, 14, 11, and 12, respectively.

From Equation 15,we have $R(Q) = \sum_{i=1}^{m}(r_i - r_{i-1})(R_i^\mu + R_i^\sigma)$, where $R_i^\mu = UB_i^\mu - LB_i^\mu$ and $R_i^\sigma = UB_i^\sigma - LB_i^\sigma$.

For R_i^μ , according to the relationship of μ_i^Q and $[\mu_i^{min}, \mu_i^{max}]$,

$$R_i^\mu = \begin{cases} (\mu_i^{max} - \mu_i^Q)^2 - (\mu_i^{min} - \mu_i^Q)^2, \text{ if } \mu_i^Q \le \mu_i^{min}; \\ (\mu_i^{max} - \mu_i^Q)^2, \text{ if } \mu_i^{min} < \mu_i^Q \le \frac{(\mu_i^{min}+\mu_i^{max})}{2}; \\ (\mu_i^{min} - \mu_i^Q)^2, \text{ if } \frac{(\mu_i^{min}+\mu_i^{max})}{2} < \mu_i^Q \le \mu_i^{max}; \\ (\mu_i^{min} - \mu_i^Q)^2 - (\mu_i^{max} - \mu_i^Q)^2, \text{ if } \mu_i^{max} < \mu_i^Q; \end{cases} \quad (16)$$

In the second and third cases, the range has the same upper bound $(\mu_i^{max} - \mu_i^{min})^2$. The more similar μ_i^{max} and μ_i^{min} are, the smaller the range is. In the first and fourth cases, it also holds that the more similar μ_i^{max} and μ_i^{min} are, the smaller the range is. Thus, we can use $(\mu_i^{max} - \mu_i^{min})^2$ to evaluate the range related to the mean value.

Similarly, for R_i^σ,

$$R_i^\sigma = \begin{cases} (\sigma_i^{max} + \sigma_i^Q)^2 - (\sigma_i^{min} - \sigma_i^Q)^2, \text{ if } \sigma_i^Q \le \sigma_i^{min}; \\ (\sigma_i^{max} + \sigma_i^Q)^2, \text{ if } \sigma_i^{min} < \sigma_i^Q \le \sigma_i^{max}; \\ (\sigma_i^{max} + \sigma_i^Q)^2 - (\sigma_i^{max} - \sigma_i^Q)^2, \text{ if } \sigma_i^{max} < \sigma_i^Q; \end{cases} \quad (17)$$

By simple transformation, we can see that, in both the first and second cases, the range is smaller than $(2\sigma_i^{max})^2$. Moreover, in the third case, the range equals to $4\sigma_i^{max}\sigma_i^Q$. In all cases, it holds that the smaller σ_i^{max} is, the smaller the range related to standard deviation is. Thus, we can use $(\sigma_i^{max})^2$ to evaluate the range related to standard deviation.

We combine the above two components and define our measurement of estimation quality as

$$Qos = \sum_{i=1}^{m}(r_i - r_{r-1})((\mu_i^{max} - \mu_i^{min})^2 + (\sigma_i^{max})^2) \quad (18)$$

The measurement Qos does not depend on query time series. The smaller Qos is, the more effective the bounds in a node are for similarity estimation.

3.3.4 *Finding Splitting Strategies*

Denote by N the node to be split, and by Qos_N its Qos value. We split N to two children nodes N_l and N_r, and denote their Qos values by Qos_l and Qos_r, respectively. We define the *splitting benefit* as $B = Qos_N - \frac{Qos_l + Qos_r}{2}$. The larger B is, the better the splitting strategy.

Now, we introduce function $BestSplit$. For each segment, we compute B for all possible vertical and horizontal splitting strategies, and select the one with the maximum B value as the best strategy. After the index building, each internal node maintains its own splitting strategy, SP. Given splitting strategy SP of node N and a query time series Q, the function $routeToChild$ can correctly find the appropriate child node. The process is similar to that of reassigning time series when splitting occurs. We first transform Q according to the segmentation of N. Then, we re-transform Q according to the corresponding splitting strategy and check which child node it belongs to, and assign Q to it.

3.4 Analysis of DSTree

In this section, we discuss some factors that are related to the performance of the DSTree index.

Adaptive segmentation. In all previous approaches, one has to specify the dimensionality of the time series representation, such as the number of coefficients in DFT and DWT, and the number of segments in iSAX and APCA. However, it is hard to determine the optimal parameters. DSTree avoids this difficulty by automatic segmentation splitting.

Data distribution. Ideally, the performance of an index is insensitive to the distribution of time series to be indexed. Many existing methods assume or target at some distributions in design. For example, the R-tree based approaches assume that all time series can be represented well using the same number of coefficients, which may not hold in many applications. If some time series are dominated by low-frequency data and the others are dominated by high-frequency data, DFT-based index may have poor performance.

DSTree does not assume any data distribution since different nodes have their own representations. For this reason, time series with dramatically different characteristics can still be handled well by DSTree using different nodes.

Balance of DSTree. iSAX and DSTree both may generate imbalanced index trees. Our experimental results (Table 2) show that

DSTree is better than iSAX 2.0 in terms of balancing because of the multiple splitting strategies.

Heuristically, we can improve the balance of DSTrees in two ways. First, we can shuffle the data set and build the index several (e.g., 3-5) times using different input orders of time series, and then pick the best one as the final index. Second, we can adjust the tree by a post-process where we move the extraordinarily deep subtrees toward the root node. Limited by space, we omit the details here.

The major search cost in DSTree and also some other time series indexes, such as iSAX, is to retrieve time series data from disk. Searching internal nodes in the index is relatively quick. Therefore, keeping similar time series in a leaf node can help to reduce the number of I/O operations needed in a similarity search. This is very different from lookup queries using B+-tree or similar indexes, where the whole index is stored on disk.

Extension to subsequence matching. The subsequence matching problem is to find matching subsequences between two time series, which may have different lengths. The state-of-the-art approaches partition (long) time series into a set of equal-length subsequences based on overlapped windows, and then build the index for these subsequences for fast similarity search. The search results are assembled to compute matching subsequences. Since the time series after partitioning for similarity search are of the same length, DSTree can be used to support subsequence matching directly.

4. QUERY ANSWERING ALGORITHMS

A DSTree supports two types of queries. The first one is the traditional similarity search, which returns the time series nearest to the query time series. The second type is to estimate the distance distribution, which returns a histogram of distances between the query time series and all indexed time series.

Algorithm 2 $exactSearch(Q)$

```
 1: Input: A query time series $Q$
 2: Output: The nearest time series $TS$ with distance $D_{bsf}$
 3: $N_{bsf} = HeuristicSearch(Q)$;
 4: $(TS, D_{bsf}) = calcMinDist(N_{bsf}, Q)$;
 5: Initialize distance priority queue $pq$;
 6: $pq.Add(N_R, D_{LB}(N_R, Q))$;
 7: while $!pq.isEmpty()$ do
 8:     $(N_{cur}, LB_{cur}) = pq.PopMin()$;
 9:     if $LB_{cur} > D_{bsf}$ then
10:         break;
11:     end if
12:     if $N_{cur}$ is a leaf node then
13:         $(X, Dist) = calcMinDist(N_{cur}, Q)$;
14:         if $Dist < D_{bsf}$ then
15:             $D_{bsf} = Dist; TS = X$;
16:         end if
17:     else
18:         for all children nodes $N'$ of $N_{cur}$ do
19:             if $D_{LB}(N', Q) < D_{bsf}$ then
20:                 $pq.add(N', D_{LB}(N', Q))$;
21:             end if
22:         end for
23:     end if
24: end while
25: return $TS$,$D_{bsf}$;
```

4.1 Similarity Search

Before introducing the exact similarity search, we first introduce a *heuristic search* method, which is more efficient and will be used in the exact search method later.

4.1.1 A Heuristic Method

Algorithm 3 $Histogram(Q)$

```
 1: Input: A query time series $Q$
 2: Output: A distance histogram $Hist$
 3: Initialize a distance range count list $L$;
 4: Initialize a node stack $Stack$;
 5: $Stack.push(Root, -\infty, +\infty)$;
 6: while $!Stack.isEmpty()$ do
 7:     $(N, LB_p, UB_p) = Stack.Pop()$;
 8:     $LB = D_{LB}(N, Q)$;
 9:     $UB = D_{UB}(N, Q)$;
10:     $LB = max(LB, LB_p)$;
11:     $UB = min(UB, UB_p)$;
12:     if $N$ is a leaf node then
13:         $Count = N.C$; $L.add(LB, UB, Count)$;
14:     else
15:         for all child node of $N$, $N'$ do
16:             $Stack.Push(N', LB, UB)$;
17:         end for
18:     end if
19: end while
20: $Hist = BuildHistogram(L)$;
21: return $Hist$;
```

Instead of finding the exact most similar time series by checking all possible nodes in a DSTree, a heuristic search only investigates one leaf node, and tries to find the most similar time series in this node. This method is based on the heuristic that similar time series are often indexed in the same node.

Specifically, given a query Q, we start from the root node. If the root node is not a leaf node, then we find a child node of the root node that can hold Q as if Q ware inserted into the index. This search process is conducted recursively until a leaf node N is met. Then, we calculate the distance $D(S, Q)$ for every time series $S \in \mathcal{TS}_N$, and return the time series of the shortest distance. Please note that the heuristic method, as the name suggests, may not find the most similar time series in the whole data set.

4.1.2 The Exact Search

To speed up search, we combine the heuristic search method and the lower bounding distance function to prune the search space. The pseudo-code is given in Algorithm 2.

The algorithm begins with a best-so-far (BSF) answer returned by the heuristic search method. The intuition is that, by quickly obtaining a time series that is likely similar to the query time series, a large portion of the search space may be pruned effectively.

Once a BSF is obtained, a priority queue, denoted by pq, is created to examine nodes that may host time series that are potentially more similar to the query time series than the BSF answer. This priority queue is initialized to include only the root node.

The algorithm then repeatedly extracts the node with the smallest lower bound distance from the priority queue until either the priority queue becomes empty or an early termination condition is met. The early termination occurs when the lower bound distance is greater than or equal to the distance of the BSF answer. When the condition is satisfied, the remaining time series in the priority queue cannot qualify as the nearest neighbor and can be pruned.

To process a node from the priority queue, two possible cases may happen. (1) In the case that the node is a leaf node, we fetch the time series from disk and compute the distance from the query to these time series, recording the minimum distance. If this distance is less than our BSF answer, we update the BSF answer. (2) In the case that the node is an internal node, its children nodes are inserted into the priority queue provided their lower bound distances to the query time series are less than the distance of the BSF answer.

4.2 Distance Distribution Histogram

Algorithm 3 gives the pseudocode of computing an equi-width histogram of the distances from a query time series to all time series indexed by a DSTree. We collect all statistical information of the leaf nodes to form a list, denoted by L, in which each entry represents the number of time series falling in certain distance range. The range can be estimated based on Theorem 2.

EXAMPLE 3. *A list* $L = \langle([10, 20], 10), ([15, 30], 15), ([40, 50], 2)\rangle$ *means that there are 3 leaf nodes:* N_1*,* N_2 *and* N_3*.* N_1 *includes 10 time series, and their distance from* Q *is between* $[10, 20]$*. The distance range and number of time series in* N_2 *and* N_3 *are* $([15, 30], 15)$ *and* $([40, 50], 2)$ *respectively.* ■

Since the entries of any two leaf nodes are disjoint, there is not redundant information in L. Thus, we can obtain a corresponding histogram quickly. One issue is that in some cases, the lower (or upper) bound of a child node may be smaller (or larger) than that of its parent node. In other words, the bounds in the parent node may be tighter than those in the children nodes. Using the bounds at such children nodes causes less accurate estimation of the children nodes. We address it with Theorem 3, which is easy to show.

THEOREM 3. *If the estimated range of the distance in a node is* $[LB, UB]$*, and that in its parent nodes is* $[LB_p, UB_p]$*, then* $[max(LB, LB_p), min(UB, UB_p)]$ *is a tighter and correct range of this node.*

Using Theorem 3, whenever a node is traversed, we first compute the lower and upper bounds according to Theorem 2. Then we compare the bounds with those in its parent node, and use the tighter bounds instead Lines 7-11 in Algorithm 3.

Furthermore, one may build a histogram more quickly by traversing some internal nodes instead of all leaf nodes. Specifically, we propose an approach here, called α-level ($0 < \alpha \leq 1$), to compute a histogram.

Denote by H the height of a DSTree. For each path from the root to a leaf node, we select the $\lceil \alpha * H \rceil$-th internal node instead of the corresponding leaf node to generate the list L. If the length of a path is shorter than $\lceil \alpha * H \rceil$, we simply use the leaf node. In other words, we use the nodes located in certain cross section of the whole tree to generate L. Algorithm 3 can be extended to this case easily. The experimental results show that we can obtain good estimation with $\frac{2}{3}$-level.

Once we obtain the list L, we can compute a histogram based on it. There are multiple ways to compute a histogram. A straightforward way is to assume that the time series contained in a node are distributed uniformly.

EXAMPLE 4. *Consider the list* L *in Example 3. We can estimate the number of time series within the range* $[15, 20]$ *as* $\frac{15-10}{20-10} * 10 + \frac{20-15}{30-15} * 15 = 10$. ■

In general, we can assume that the time series in a node follow some distribution, such as Gaussian distribution. We can spend some extra space to maintain the parameters of the model, such as mean and variance, which allow more accurate and efficient estimation. Limited by space, we omit the details here.

5. EMPIRICAL EVALUATION

In this section, we report extensive experiments to verify the effectiveness of DSTree. We compare both PAA-index (using PAA as representation and R-tree as index) and iSAX2.0 with DStree in index efficiency, approximate search error rate and pruning power. We also showcase the lower bound tightness and accuracy of histogram estimation. All experiments were executed on a laptop computer with an Intel Core i5 2.5GHz CPU and 4GB main memory. All experimental results were averaged over 50 runs.

5.1 Data Sets and Default Setting

The time series in both synthetic and real data sets were normalized with Z-normalization.

5.1.1 Synthetic Data sets

Each of our synthetic data set is a combination of four types of time series as follows.

- Random walk times series. The start point is picked randomly from range $[-5, 5]$ and the step length is chosen randomly in range $[0, 2]$;
- One-segment Gaussian time series. The values in the whole time series are picked from a Gaussian Distribution with mean value and standard deviation randomly selected in ranges $[-5, 5]$ and $[0, 2]$, respectively;
- Multi-segment Gaussian time series. Such a time series is concatenated by multiple one-segment Gaussian time series. The number of segments is randomly set between 3 to 10.
- A mixed sine time series. Each time series is a mixture of several sine waves whose period is randomly set in range $[2, 10]$, amplitude is randomly set in range $[2, 10]$, and mean value randomly chosen in range $[-5, 5]$.

To generate a time series, the synthetic data generator first randomly chooses a type, and then picks the corresponding parameters randomly to generate the time series. We generated four synthetic data sets of time series lengths 64, 128, 256 and 512, respectively. Each data set contains one million time series by default. We also use synthetic data sets of up to 200 million time series in the scalability test.

5.1.2 Real Data sets

We used a real data set collected in a bridge condition monitoring system. In this system, data was collected from about one thousand sensors of more than 20 types, such as thermometers, accelerometers, strain gauges, displacement meters, and fatigue meters. The length of each time series is 256, and one million times series were collected. The total storage space is about 3GB.

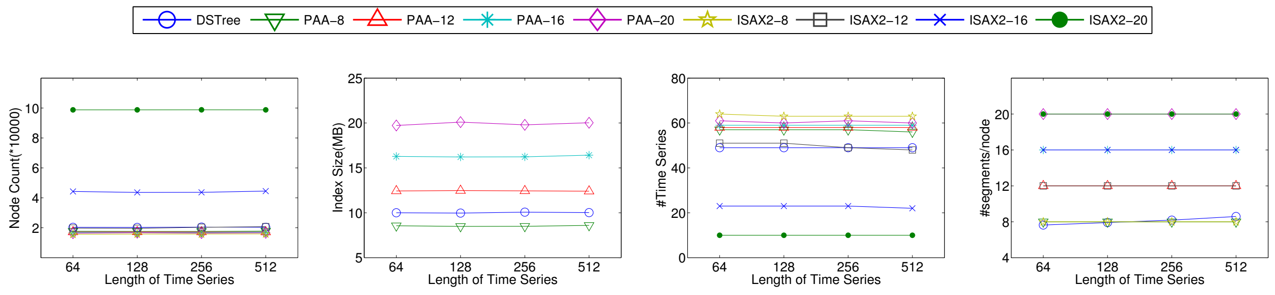
5.1.3 Parameters

To verify the effectiveness of data-adaptive and dynamic segmentation versus global segmentation, we compared DSTree with PAA-index (implemented by ourselves) and iSAX2.0 (source code provided by the authors). Both PAA-index and iSAX2.0 use fixed, global segmentations. To test the performance extensively, we built PAA-index and iSAX2.0 with segment sizes of 8, 12, 16 and 20 respectively. The leaf capacity threshold, ψ, was set to 100. The FBL size for iSAX2.0 was set to 200,000. The fill factor of R-tree in PAA-index was set to 0.5.

5.2 Index Size

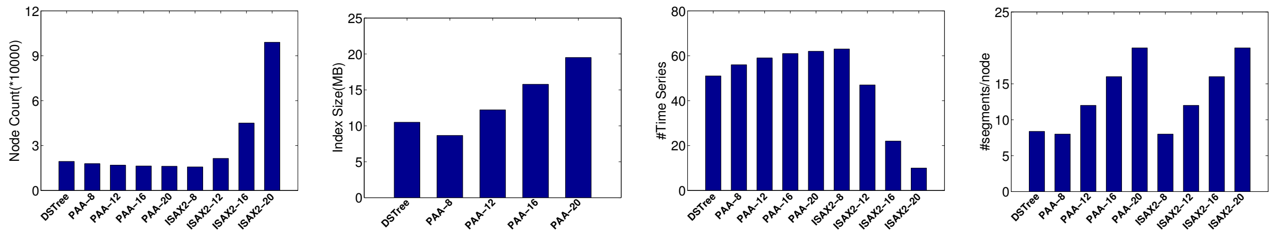
We did not implement iSAX2.0 by ourselves. Instead, we used the implementation provided by the authors. We realize that the the implementation details, particularly the storage methods, in iSAX2.0 and DSTree may be different. To avoid any confusion, we report the absolute index size for the methods we implemented but not for iSAX2.0.

The first group of experiments compare the index space cost of DSTree, PAA-index and iSAX2.0 with respect to the length of time series. Specifically, we report three measurements, namely the number of nodes in the tree, the physical index size, and the average number of time series contained by a leaf node. The number of nodes includes both internal and leaf nodes. Considering the difference on data representations in the three approaches, we also compare the physical index size for DSTree and PAA. We use the average number of time series in leaf nodes to evaluate the balance



(a) Number of nodes (b) Index size (c) Average #ts per leaf (d) # segments/node

Figure 5: Index size on the synthetic data sets.



(a) Number of nodes (b) Index size (c) Average #ts per leaf (d) # segments/node

Figure 6: Index size on the real data set.

of the index nodes. The results on the synthetic data sets are shown in Figure 5, and those on the real data set are in Figure 6. Four different segmentation sizes, 8, 12, 16 and 20, were tested for both PAA-index and iSAX2.0. Label "PAA-16" means PAA-index with 16 segments.

Figure 5(a) shows that, in all the three approaches, the number of nodes is insensitive to the length of time series. However, the number of segments has different effects on iSAX2.0 and PAA-index. In iSAX2.0, the number of nodes increases exponentially as the number of segments increases, for example, iSAX2-16 and iSAX2-20 have much more nodes. PAA-index is insensitive to the number of segments. The number of nodes in DSTree is stable and far less than those in iSAX2-16 and iSAX2-20. The number of nodes affects the search efficiency. If it is too large, the average number of time series per leaf node decreases and more I/O cost is needed to read data from disk.

Figure 5(b) compares the absolute index size, the smaller, the better. The size of an index is determined by two factors: the number of nodes and the unit space cost per node. For PAA-index, the space cost of each node increases almost linearly as the number of segments increases. Since DSTree needs to maintain both mean and standard deviation values, it has a larger unit space cost. However, benefitting from the dynamic splitting strategies, the average segment size of DSTree is small.

Figure 5(c) shows the average number of time series per leaf node. The smaller the number, the fewer time series in expectation can be retrieved from a leaf node. The number in DSTree is about 50. The number in PAA-index is the largest (about 60) due to the R-tree structure. In iSAX 2.0, this value decreases when the number of segments increases for two reasons. First, in iSAX2.0, the root node has too many children nodes (for example, 2^{16} for iSAX2-16). Second, it uses fixed segmentation. In some cases, it is difficult to split a set of time series only based on the mean value.

Since DSTree uses a dynamic segmentation strategy, the segment size varies in different nodes. We report the average segment size with respect to length of time series, that is, the ratio of the total number of segments in all nodes against the number of nodes. Figure 5(d) shows the results. The average segment size of DSTree increases moderately when the length of time series increases, which confirms the effectiveness of our splitting strategies. The average segment sizes of the other approaches are insensitive to the length of time series.

Figure 6 shows the results on the real data set. The trends are similar to those on the synthetic data sets. The number of nodes of DSTree is similar to those of iSAX2-12 and smaller than those of iSAX2-16 and iSAX2-20. The average number of segments per node of DSTree is 8.38. Although the time series in the real data set are more diverse, DSTree can still represent the time series with a small number of segments, which verifies the effectiveness of the dynamic splitting strategy in DSTree.

Both iSAX2.0 and DSTree are binary trees. To examine the balance of those indexes, Table 2 compares the average height of iSAX2.0 and DSTree. We use the normalized standard deviation (that is, standard deviation divided by the average) of the tree height to measure the balance of the trees. We do not consider PAA-index in this comparison because R-tree, though balanced, has a much larger fan-out factor.

The height of all indexes increases very moderately as the time series length increases. These indexes are all scalable with respect to long time series. iSAX2.0 are substantially shorter than DSTree in average height, but clearly taller than DSTree in maximum height. The dynamic splitting strategy in DSTree can effectively avoid long branches. The small normalized standard deviation values in DSTree clearly show that DSTree has good balance.

Table 3 examines the effect of the leaf capacity threshold ψ. The number of nodes and index size of DSTree, iSAX2-12, and iSAX2-8 decrease dramatically as ψ increases. iSAX2-16 and iSAX2-20 do not gain much from a larger leaf capacity threshold. One reason is that iSAX2.0 with m segments may have up to 2^m children nodes of the root node, though many such nodes may contain a very small number of time series.

5.3 Accuracy

We tested the effectiveness of the indexes in similarity search,

Data set	DSTree			iSAX2-8			iSAX2-12			iSAX2-16			iSAX2-20		
	Avg	NSD	Max	Avg	NSD	Max	Avg	NSD	Max	Avg	NSD	Max	Avg	NSD	Max
S64	18.09	0.26	29	11.40	0.39	44	7.38	0.59	38	2.58	1.25	36	1.57	1.13	36
S128	18.44	0.26	30	11.60	0.38	42	7.56	0.59	39	2.67	1.25	32	1.63	1.17	36
S256	18.57	0.28	29	11.60	0.39	44	7.63	0.59	41	2.73	1.26	36	1.69	1.22	37
S512	18.96	0.29	30	11.74	0.39	43	7.70	0.60	42	2.81	1.26	38	1.74	1.21	34
R256	18.67	0.29	30	11.53	0.39	35	7.61	0.59	35	2.69	1.26	32	1.67	1.21	30

Table 2: Average height (Avg), normalized standard deviation (NSD), and maximum length (Max) of the indexes. ("S256" denotes the synthetic data set with the length of time series 256, and "R256" denotes the real data set with length 256.)

Leaf capacity threshold ψ	DSTree		iSAX2-8	iSAX2-12	iSAX2-16	iSAX2-20
	# nodes	Size (MB)	# nodes	# nodes	# nodes	# nodes
100	19338	10.07	15632	19754	44244	98796
1000	2163	0.99	1784	5003	36592	93356
2000	1196	0.56	957	4431	36325	93191
5000	485	0.21	452	4177	36196	93121

Table 3: Number of nodes and index size (MB) versus leaf capacity threshold ψ.

including both heuristic search and exact search. The accuracy of heuristic search is measured by the error rate $E = \frac{|\overline{D}-D|}{D}$, where D and $\overline{D}$ are the distance between the query time series and the exact nearest neighbor and the heuristic search result, respectively.

For exact search, we compare the *pruning power*, which is the ratio of the number of time series pruned against the total number of time series. For both heuristic and exact search, 100 time series were used as the queries, half of them picked randomly from the data set, and the rest generated randomly. Figures 7 and 8, respectively, show the results on the synthetic and real data sets.

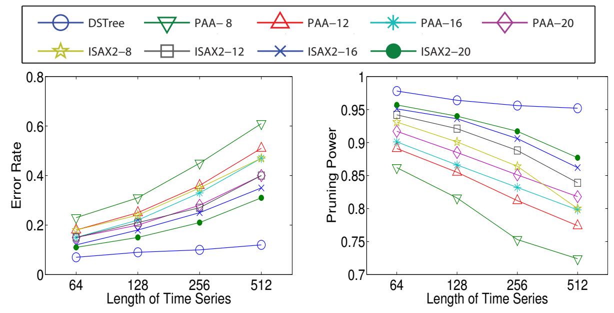
In Figure 7(a), although the error rate increases as the length of time series increases for all three methods, DSTree outperforms the others clearly. In Figure 8(a), the error rate decreases for PAA-index and iSAX2.0 when the size of segment increases, since using more segments can represent the time series more accurately. With the same segment size, iSAX2.0 outperforms PAA-index. Interestingly, when the query time series is picked from the data sets, both iSAX2.0 and DSTree correctly find the leaf node containing the right time series due to the disjoint space division property of these two approaches. PAA-index finds the wrong node in some of such cases, because of the intersection of MBR in R-tree. When the query is generated randomly, DSTree is more accurate in finding the corresponding leaf nodes than iSAX2.0 because of our data-adaptive splitting strategy.

Figures 7(b) and 8(b) show the pruning power of exact similarity search. The pruning power of DSTree is greater than 95% on all synthetic data sets and is 98% on the real data set, which is clearly better than those of the other two approaches. The pruning power of PAA-index increases dramatically as the segment size increases from 8 to 20. However, the marginal performance gain decreases as the segment size increases further. A reason is that R-tree performs poorly with high dimensionality. iSAX2.0 has a similar trend.

The advantages of DSTree are from two factors. First, a tighter lower bound helps to prune more nodes. Second and more importantly, the proposed data adaptive splitting strategies can cluster similar time series better. Consequently, the heuristic search in DSTree is more accurate, which gives DSTree a good starting point in exact search. Moreover, fewer data files are visited since similar time series are clustered better into fewer nodes.

5.4 Lower Bound Tightness

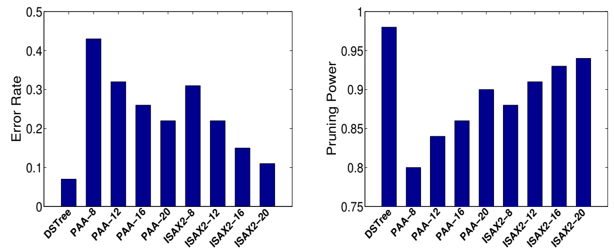
We tested the tightness of the proposed lower bound estimation approach. We measure the *lower bound tightness* by the ratio of the estimated lower bound distance against the minimum distance from a query to all time series indexed in a node. This ratio is between



(a) Heuristic search error rate

(b) Exact search pruning power

Figure 7: Error rate and pruning power on the synthetic data sets.



(a) Heuristic search error rate

(b) Exact search pruning power

Figure 8: Error rate and pruning power on the real data set.

0 and 1, the larger, the better. We collected this information during the processing of exact search. Figure 9 shows the results on both the synthetic and real data sets.

The lower bound using both mean and standard deviation values is tighter than that using only mean values.

5.5 Histogram Computation

Figure 10 compares the exact distance histogram by a full scan of the data and the distance histogram estimated the α-level method (Section 4.2) . For the latter, three cases are shown. "Full level" uses all leaf nodes to estimate. "2/3 level" and "1/3 level", respectively, use internal nodes located at the 2/3-level and 1/3-level cross sections to compute the histogram. Although the full and 2/3

# time series (millions)	DSTree # nodes	DSTree Size (MB)	iSAX2-8 # nodes	iSAX2-12 # nodes	iSAX2-16 # nodes	iSAX2-20 # nodes
10	4,335	2.43	3,643	5,966	59,676	235,558
50	20,576	12.88	16,394	20,642	77,579	287,904
100	38,559	27.31	30,985	38,957	93,095	340,250
200	69,987	42.57	57,239	59,801	113,404	392,597

Table 4: Number of nodes and index size (MB) vs. #time series

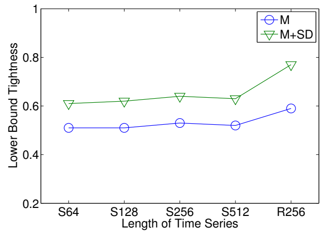


Figure 9: Lower bound tightness (M: lower bounds using only mean values; M+SD: lower bounds using both mean and standard deviation values.)

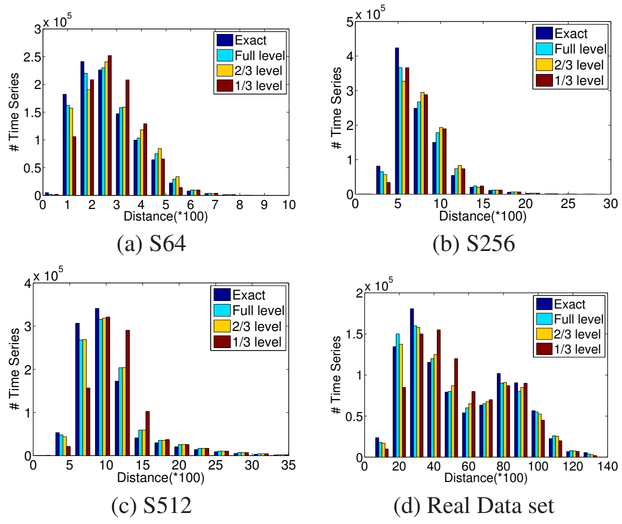


Figure 10: Histogram of distance distribution.

level estimation do not access the time series data and build the histogram by only merging list L computed from the index, the obtained histograms are still very close to the exact one. Even the 1/3 level estimation gives an acceptable estimation.

The α-level method saves substantial time in histogram estimation. For example, on the data set S512, a full scan to compute the exact histogram takes 161 seconds, while the α-level method takes only 115 ms, 84 ms, and 8 ms, respectively, when α is set to 1, $2/3$, and $1/3$.

5.6 Scalability

To test the scalability of DSTree and iSAX2.0, we use the data generator provided by iSAX2.0 to generate 4 data sets containing 10 million, 50 million, 100 million and 200 million time series, respectively. Each time series is of length 256. In all experiments, the leaf capacity ψ is set to 5,000. The results of the number of nodes and absolute index size are shown in Table 4, and the results of pruning power and index building time are shown in Figure 11.

As shown in Table 4, the number of nodes of DSTree is similar to those of iSAX2-8 and iSAX2-12, and much less than those of iSAX2-16 and iSAX2-20. Figure 11(a) shows that DSTree has a higher pruning power than all iSAX2.0 indexes. Moreover, in all the methods, the pruning power is larger on bigger data sets. When a data set has more time series, the time series in each leaf node are more similar. Consequently, more irrelevant time series can be pruned, and fewer data files are needed to visit to find the most similar time series.

Figure 11(b) shows that the building time of all indexes are roughly linear to the data set size. The building time of DSTree is a little longer than that of iSAX2-8 and iSAX2-12, and clearly shorter than that of iSAX2-16 and iSAX2-20.

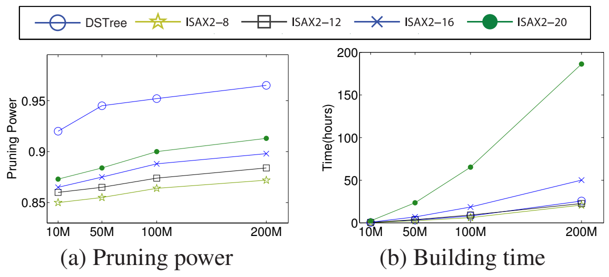


Figure 11: Scalability of the indexes.

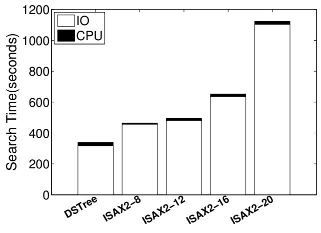


Figure 12: Searching time (200 million time series, length=256, $\psi = 5,000$)

5.7 Search Efficiency

To compare the search efficiency of iSAX2.0 and DSTree, we used the synthetic data set of 200 million time series of length 256 each, which is used in the scalability test. The leaf capacity was set to 5,000. 100 time series were used as the queries, half of which were picked randomly from the data set, and the rest were generated randomly. For iSAX2.0, the number of segments were set to 8, 12, 16 and 20. The results are shown in Figure 12. Since the index size is not large (less than 100 MB), we hold the whole index in the main memory during searching.

It can be seen that the search time of DSTree is less than those of all iSAX index. We investigate the reason by analyzing the components of the search time in more detail. In general, the search time is composed of three parts: time to traverse the index, time to read data files from disk and time to compare query and target time series, in which, the first and third ones are related to CPU computing, and the second related to disk I/O. We also distinguish

CPU related time and I/O related time in Figure 12.

In all methods, the CPU cost is only less than 10% of the total search time, and the I/O cost dominates the search time. The I/O efficiency is affected by two factors: the pruning power and the number of nodes. A high pruning power and a small number of nodes can reduce the number of time series that have to be retrieved from disk. DSTree achieves the best search efficiency overall.

6. RELATED WORK

Many studies on similarity search over time-series databases have been conducted in the past decade. The pioneering work by Agrawal *et al.* [1] used Euclidean distance as the similarity measure, Discrete Fourier Transform (DFT) as the dimensionality reduction tool, and R-tree [7] as the underlying search index. Faloutsos *et al.* [6] later allowed subsequence matching and proposed the FRM framework for indexing time series.

The subsequent work focused on two major aspects: new dimensionality reduction techniques (assuming that Euclidean distance is the underlying measure) and index building techniques based on dimensionality reduction techniques. Existing dimensionality reduction techniques include SVD [8], DFT [1], DWT [4], PLA [14], PAA [21], APCA [12] and CP [2]. These methods first reduce the dimensionality of each time series to a lower dimensional space, and then apply a new metric distance function to measure the similarity between any two transformed (reduced) time series. In order to guarantee no-false-dismissals during the similarity search, the metric distance function must satisfy the lower bounding lemma [6].

Among all the reduction methods, SVD is accurate, but, at the same time, costly in both time and space, since SVD needs to calculate eigenvectors and store large matrices using extra space. Furthermore, APCA [12] and CP [2] are the two state-of-the-art reduction approaches. Keogh *et al.* [12] indicated that APCA outperforms DFT, DWT, and PAA in terms of the pruning power by orders of magnitude.

Approaches to building indexes can be categorized into two types. First, the traditional multi-dimensional index approaches, like R-tree, are used without modification [1, 11, 6]. Second, the R-tree method is modified according to the representation of time series [12, 5]. Since APCA contains both mean values and right ending points in the approximate representation, Keogh *et al.* [12] redefined the MBR (Minimal Bound Rectangle) according to APCA. PLA represents time series by disjoint lines. Each line is represented by two parameters: slope and intercept. Chen *et al.* [5] proposed a new MBR definition accordingly.

Most of the previous index approaches can be split into two phases: dimension reduction first and then building the index. Representing time series approximately is independent from building the index. Most recently, a new family of index approaches, iSAX [16] and iSAX 2.0 [3], were proposed based on the representation technique, named SAX, which is a symbolic representation for time series that allows dimensionality reduction and indexing with a lower bounding distance measure. To the best of our knowledge, iSAX and iSAX 2.0 are the only approaches that transform time series during index building, and are data adaptive. Particularly, iSAX is based on SAX representation of time series. It segments all time series into equi-length segments, and symbolizes the mean value of each segment. The symbolization is executed during the index building, which makes iSAX a data-adaptive approach. However, the fixed number of segments and the fixed length of segments constrain their capability of being fully data-adaptive.

Some existing methods use upper and lower bounds. For example, Vlachos *et al.* [17] proposed optimal lower and upper bounds of distance between two time series, which are based on Discrete Fourier Transform. Karamitopoulos and Evangelidis [9] extended APCA by using standard deviation to define the lower bound. Wei *et al.* [19] bounded the distance between a query time series and a set of time series. Our approach differs from those methods in several aspects. First, in a DSTree different nodes may use different numbers of segments to represent time series, while all previous approaches use a fixed number of dimensions. Second, in a DSTree different nodes may use different splitting strategies. Moreover, the previous methods do not define an upper bound by extending APCA. We combine lower bound and upper bound to measure the quality of splitting.

7. CONCLUSIONS

In this paper, we made a critical observation: global segmentation of time series may lead to unnecessary cost in space and time for indexing. To tackle the problem, we developed a data-adaptive and dynamic segmentation index, DSTree, which saves space and time in indexing, and achieves tighter upper and lower bounds on distance estimation. We presented an extensive performance study on synthetic and real data sets to verify the effectiveness and efficiency of our new approach in both similarity search and histogram estimation.

8. REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *FODO*, pages 69–84, 1993.

[2] Y. Cai and R. Ng. Indexing spatio-temporal trajectories with chebyshev polynomials. In *SIGMOD*, pages 599–610, 2004.

[3] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. isax 2.0: Indexing and mining one billion time series. In *ICDM*, pages 58–67, 2010.

[4] K. Chan and A. Fu. Efficient time series matching by wavelets. In *ICDE*, pages 126–133, 1999.

[5] Q. Chen, L. Chen, X. Lian, Y. Liu, and J. Yu. Indexable pla for efficient similarity search. In *VLDB*, pages 435–446, 2007.

[6] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, 1994.

[7] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, 1984.

[8] K. Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *SIGMOD*, pages 166–176, 1998.

[9] L. Karamitopoulos and G. Evangelidis. A dispersion-based paa representation for time series. In *CSIE*, pages 490–494, 2009.

[10] E. Keogh. A decade of progress in indexing and mining large time series databases. In *VLDB*, page 1268, 2006.

[11] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. In *KAIS*, pages 263–286, 2000.

[12] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD*, pages 151–162, 2001.

[13] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. In *SIGKDD*, pages 102–111, 2002.

[14] E. Keogh and M. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *SIGKDD*, 1998.

[15] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *DMKD*, pages 2–11, 2003.

[16] J. Shieh and E. Keogh. isax: Indexing and mining terabyte sized time series. In *SIGKDD*, pages 623–631, 2008.

[17] M. Vlachos, S. S. Kozat, and P. S. Yu. Optimal distance bounds on time-series data. In *SDM*, pages 109–120, 2009.

[18] L. Wei and E. Keogh. Semi-supervised time series classification. In *SIGKDD*, pages 748–753, 2006.

[19] L. Wei, E. Keogh, H. Van Herle, and A. Mafra-Neto. Atomic wedgie: Efficient query filtering for streaming time series. In *ICDM*, pages 490–497, 2005.

[20] X. Xi, E. Keogh, C. Shelton, L. Wei, and C. Ratanamahatana. Fast time series classification using numerosity reduction. In *ICML*, pages 1033–1040, 2006.

[21] B. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *VLDB*, pages 385–394, 2000.