The TS-Tree: Efficient Time Series Search and Retrieval

Ira Assent, Ralph Krieger, Farzad Afschari, Thomas Seidl
Data Management and Exploration Group
RWTH Aachen University, Germany
{assent,krieger,afschari,seidl}@cs.rwth-aachen.de

ABSTRACT

Continuous growth in sensor data and other temporal data increases the importance of retrieval and similarity search in time series data. Efficient time series query processing is crucial for interactive applications. Existing multidimensional indexes like the R-tree provide efficient querying for only relatively few dimensions. Time series are typically long which corresponds to extremely high dimensional data in multidimensional indexes. Due to massive overlap of index descriptors, multidimensional indexes degenerate for high dimensions and access the entire data by random I/O. Consequently, the efficiency benefits of indexing are lost.

In this paper, we propose the TS-tree (time series tree), an index structure for efficient time series retrieval and similarity search. Exploiting inherent properties of time series quantization and dimensionality reduction, the TS-tree indexes high-dimensional data in an overlap-free manner. During query processing, powerful pruning via quantized separator and meta data information greatly reduces the number of pages which have to be accessed, resulting in substantial speed-up. In thorough experiments on synthetic and real world time series data we demonstrate that our TS-tree outperforms existing approaches like the R*-tree or the quantized A-tree.

1. INTRODUCTION

Many applications in science, e-commerce and surveillance monitoring rely on recording sensor information in regular time intervals. As a consequence, time series data has been growing tremendously. This flood of data requires data management for fast and easy storage and access. Similarity search on time series data is a typical requirement for such data bases. Similar patterns in sensor data might indicate a common cause or provide a prediction for future sensor values.

To enable efficient similarity search, R-trees and other multidimensional indexing structures have been used [16]. These multipurpose indexing structures, however, are designed for relatively low-dimensional data. For the special case of time series data, they are usually not adequate. Most time series are long, which corresponds to many dimensions in multidimensional indexes. Multidimensional indexes, however, are known to provide efficiency gains only up to a certain dimensionality [28]. In the case of the R-tree, sequential scan is faster from about 16 dimensions, depending on the application [5]. One of the reasons why R-trees will eventually fail is that in high-dimensional spaces MBRs (minimum bounding rectangles) of the subtrees overlap to a high degree. Therefore more paths have to be accessed in query processing, requiring expensive random reads of index and data pages.

In time series data, this problem is often aggravated. Summarizing data in minimum bounding regions, the sequential nature of time series is not captured. For example, in temperature measurements, successive values are typically within a range of few degrees. Consequently, overlap in these dense areas is large, resulting in voluminous MBRs with large proportions of empty space. As typical queries intersect with many of these MBRs, the number of excess page reads increases further. Consequently, these indexes degenerate to random read of the entire data.

B-trees are another family of indexes that have been used for sequences e.g. in the Prefix B-tree, with further compression of blocks via Patricia tries in the String B-tree [4, 14]. Prefixes are used as separators between subtrees. Since only the prefix is actually used to index the data this leads to poor pruning power for similarity search on long time series data. In turn, many paths have to be accessed as well, negating efficiency gains.

To formalize similarity such that sensor data can be searched accordingly, L_p norms and Dynamic Time Warping (DTW) are the most common models. L_p norms, such as Euclidean distance, compare time series one position at a time and have been shown to work well in a number of applications. To allow matching of slight shifts in time series, DTW stretches or squeezes the time series along the time axis to align their value patterns. The remaining differences between these best matches constitute the distance [8]. For indexing, envelope-style upper and lower bounds have been proposed [16]. They allow for exact indexing of time series at the expense of additional relevant regions. Therefore, DTW-based similarity search typically entails more page reads.

In this paper, we propose a novel index structure, the TS-tree (time series tree). It is specialized for efficient similarity search on time series. The TS-tree avoids overlap and provides compact meta data information on the subtress, thus reducing the search space very effectively. To ensure high fanout, which in turn results in small and efficient trees, index entries are quantized and dimensionality reduced. Depending on the nature of the time series, e.g. smooth or bursty, quantization brings out the relevant characteristics via SAX or wavelet analysis, respectively [20, 26]. For powerful pruning, we provide a formal $mindist$ function between any query time series and the entire available meta data. This $mindist$ enables efficient query processing without false dismissals.

Our contributions include

- a novel overlap-free index structure for time series
- efficient query processing on highly descriptive meta data
- support for the most common time series similarity models

Our paper is structured as follows: we review related work in Section 2. An analysis of inherent properties of time series and indexing paradigms is presented in Section 3. We define and discuss our novel TS-tree in Section 3.2. Query processing techniques and algorithms are discussed in Section 5. We demonstrate substantial efficiency gains in the Experiments, Section 6. We conclude in Section 7.

2. RELATED WORK

For efficient and effective time series indexing, multistep filter-and-refine approaches have been proposed, as discussed in Section 5.3. Due to the length of time series, dimensionality reduction is typically required. Time series may contain hundreds of values which hinders applicability of most indexing structures as is. Piecewise aggregate approximation (PAA) and symbolic aggregate approximation (SAX) [18, 29, 20] outperform other dimensionality reduction techniques like singular value secomposition or discrete fourier transform (SVD, DFT) for time series data [26].

B-trees [2, 3], on which most hierarchical indexing structures are based, were originally developed for one-dimensional data. They have been adapted for string data in [4, 14]. B-trees use prefix separators, thus no overlap for unique data objects is guaranteed. We analyze this technique for time series in Section 3.2.

Multidimensional indexing structures like the R-tree or R*-tree organize data in minimum bounding rectangles (MBR). This approach works well for spatial or point data in lower dimensional settings. For high dimensionality, R-trees and their variants fall prey to the curse of dimensionality [9, 10]. We discuss this approach for time series in Section 3.2.

Specialized indexing structures for high dimensions have been proposed. The X-tree (extended node tree), for example, uses a different split strategy to reduce overlap [7]. When low-overlap split is impossible, supernodes of double size are created, eventually degenerating into sequential scan. Our analysis of the disadvantages of MBR-structure in R-trees in principle generalizes to X-trees. The A-tree (approximation tree) uses VA-file-style (vector approximation file) quantization of the data space to store both MBR and VBR (virtual bounding rectangle) lower and upper bounds [6, 24, 28]. While VBRs provide additional pruning information about the subtrees, the overhead of additional meta data reduces fanout. For time series, both MBR and VBR information suffer from the drawbacks as in R-trees. As the VBRs are quantized lower and upper bounds with respect to the MBRs, problems with overlap persist.

The TV-tree (telescopic vector tree) is an extension of the R-tree [21]. It uses minimum bounding regions (spheres, rectangles or diamonds, depending on the type of L_p norm used) restricted to a subset of active dimensions. Active dimensions are the first few dimensions which allow for discrimination between subtrees. As the dimensionality in index nodes is reduced, fanout increases. However, overlap is still high, especially in the dimensions not considered during query processing. For similarity search, non-active dimensions have to assumed as infinite, resulting in poor pruning power.

The most popular similarity models for time series are L_p norms (typically Euclidean distance) and dynamic time warping (DTW). L_p norms compare values of corresponding points in time. DTW stretches or squeezes the time axis to compute the best match [23]. Using an envelope of upper and lower bounds around time series, indexing of short (about 16 dimensions) time series is efficient [16, 30]. Other similarity models, e.g. for query log analysis are discussed in [27, 12, 22].

3. TIME SERIES SIMILARITY SEARCH

Sensor data or any other type of measurements result in sequences of values called "time series". Examples include stock data, temperature measurements and network traffic volume. Most time series are periodical recordings. This means that subsequent values are typically measured at regular intervals e.g. on a daily or hourly basis.

3.1 Time series

Time series are sequences of values ordered with respect to the time associated with the values. Formally:

Definition 1. ***Time series.***
A time series t is a temporally ordered sequence of values

$$t = (t_1, \ldots, t_n), t_i \in \mathbb{R},$$

where time point $f(i)$ is before $f(i+1)$: $f(i) < f(i+1)$, and $f : \mathbb{N} \rightarrow \mathbb{R}$ is a function mapping indices to time points.

Typically, time series are very long. Consider daily stock data measurements for periods of several years or even hourly temperature measurements for decades. Other applications of sensor based monitoring might record important values per hour or per minute. This is an important property which distinguishes time series data from many other multimedia data. Long time series data which is treated as point data, corresponds to very high dimensional feature spaces. This is a challenge for similarity search and indexing, as we will discuss later on.

In many applications, time series are *smooth*, i.e. subsequent values are within predictable ranges of one another [26]. For example, stock data typically exhibits only small changes within a few percentage points per day. Similarly, temperatures in a climate monitoring application do not jump from -20°C to +20°C. Rather, if one day's temperature is 10°C, we would expect the following day to have a temperature only a few degrees higher or lower. Thus, most time series data exhibits a correlation between subsequent values. This correlation is often used to mimic time series by synthetic random walk sequences [11, 1].

3.2 Core concept of the TS-tree

The design goal of the TS-tree is to create a compact yet detailed index structure for time series similarity search and retrieval. Compactness means that similar time series should be located in the same subtree using minimal storage space and directory overhead in overlap-free subtrees. Detailed information is necessary for powerful pruning during similarity search. The TS-tree exploits the inherent properties of time series as mentioned above. We describe the core concepts in this section before giving formal definitions in the next.

The TS-tree core concepts:

- **Hierarchical tree structure:** provides directory information on several levels for subtree data. As discussed above, we have to avoid degeneration into random read of the entire data base. Existing multidimensional indexing structure do not scale to high dimensions [9, 28, 10] and produce largely overlapping descriptors. We therefore focus on compact, overlap-free descriptors for time series that provide substantial pruning power.

- **Large capacity:** is necessary for small trees with high fanout. Dimensionality reduction substantially shortens time series, requiring less memory while keeping the most relevant information. Similarly, quantization of the exact time series values to discrete symbols increases capacity of index nodes.
- **Overlap-free compact subtrees:** Like R-trees or B-trees, the TS-tree is a balanced tree that grows in a bottom-up fashion. Time series inserts into leaf nodes may lead to overflow, entailing node splits that may propagate up the tree. Splitting strategy is crucial for trees as it affects the compactness and utilization of the entire index. Most notably, overlap-free splitting is a major concern as time series are high-dimensional, which leads to degenerative overlap in R-tree family indexes.
  - **Overlap-free.** Overlap-free indexing is possible by exploiting lexicographic ordering on time series. Developed originally for one-dimensional data in B-trees, TS-tree *separators* are time series prefixes [2]. These prefixes separate time series smaller than the separator from large ones with respect to lexicographic order. Smaller time series are located in subtrees to the left of the separator, larger ones in subtrees to its right. This is unambiguous as lexicographic order is a total order. Splitting in lexicographic manner thus ensures overlap-free subtrees TS-tree descriptors.
  - **Compact.** Obviously, for real values common prefixes are rare. Short separators in the first few dimensions provide little information. While this is desirable for large fan-outs, pruning power is poor. *Rough quantization* of separators using discretization to very few symbols yields longer separators which actually provide more information in more dimensions. Moreover, rough quantization maps similar time series to the same quantized representation. This means that they are very likely stored in the same subtrees, resulting in compact TS-trees.
  - **Descriptive.** Separators split one dimension at a time in lexicographic order. The TS-tree exploits the *order of dimensionality reduction* through DWT (discrete wavelet transform) or PCA (principle components analysis). Ordering is according to degree of information in dimensions, measured as variance or level of detail. In TS-tree this ordering means that dimensions with more descriptive content are split first, leading to descriptive separators.
- **Overlap-free detailed subtrees:** during query processing, long separators provide meaningful information for powerful pruning on the dimensions covered by the separators. More detailed information on those dimensions that have not (yet) been split, is provided by additional, and more finely quantized, *meta data* in TS-trees. They consist of upper and lower bounds per dimension of the time series values in the corresponding subtree, like MBRs (minimum bounding rectangles) as used in R-tree family indexes [15]. In TS-trees, they provide additional descriptor information for query processing. Note that subtrees are still *overlap-free*, as we keep separator split and do not adopt MBR-split.

In summary, TS-trees exploit the inherent properties of time series. Lexicographic separator split in coarsely quantized time series ordered with respect to the most descriptive reduced dimensions are coupled with finely quantized meta data information for powerful pruning. This allows for efficient and effective similarity queries for time series data. In the next section, we detail the structure of TS-trees in a formal manner before proceeding to query processing on the complex descriptor information.

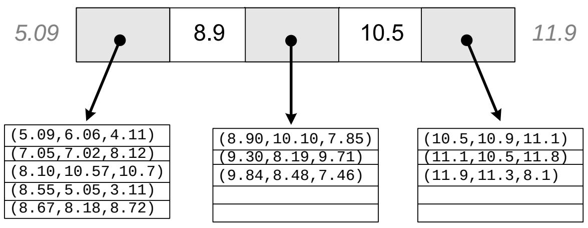


Figure 1: Separators

4. THE TS-TREE

In this section, we formalize the TS-tree. It is a hierarchical structure which extends the node structure of R-trees or B-trees and uses B-tree split to ensure balancing. The descriptors stored in the TS-tree nodes are lower and upper bounds of the dimensionality reduced time series data as well as quantized separators between subtrees. Separators are prefixes of time series, typically much shorter than the entries in data leafs, that are lexicographically larger than subtrees to the left and smaller than subtrees to the right:

DEFINITION 2. ***Separator.***
A separator $\mathbf{S}$ *between two time series* t_l *and* t_r *is a time series with:*

- $t_l \leq \mathbf{S}$ *(lexicographically larger than left time series)*
- $\mathbf{S} \leq t_r$ *(lexicographically smaller than right time series)*
- $\nexists \mathbf{S}' : |\mathbf{S}'| \leq |\mathbf{S}| \wedge t_l \leq \mathbf{S}' \leq t_r$ *(as short as possible)*

where lexicographically smaller is defined as: $a \leq b$ *iff*
$(\exists j \leq \min\{|a|, |b|\}\, \forall i \in \{1, \ldots, j-1\} : a_i = b_i \wedge a_j < b_j) \vee$
$(|a| \leq |b| \wedge\ \forall i \in \{1, \ldots, |a|\} : a_i = b_i)$

A separator is thus the shortest possible time series that differentiates between time series to the left and to the right in nodes. Lexicographic ordering is the numeric order in the first dimension in which time series differ (if any, else the shorter one is smaller). Separators are illustrated in Figure 1. The root node contains the one-dimensional time series separators 8.9 and 10.5, the left subtree's sequences all begin with values smaller than 8.9, the sequences to the right start with larger values. All elements in the middle subtree are time series which begin with a value between 8.9 and 10.5. As we can see from this small example, for continuous values, common prefixes are rare and separators are extremely short. Thus, there is only information on the very first dimension's value ranges. Comparing a query time series against short separators leads to very low values that are typically not sufficient for pruning.

The TS-tree uses quantization to group similar values. Mapping similar values to discrete symbols, common prefixes are more likely to occur. Consequently, separators are enlarged and pruning power is enhanced. Quantization is defined as any mapping of continuous time series values to time series of discrete symbols:

DEFINITION 3. ***Quantization.***
A time series $q = (q_1, \ldots, q_n)$ *is a quantization of a time series* $t = (t_1, \ldots, t_n)$ *with respect to a set of symbols* $\{s_1, \ldots, s_k\}$ *iff*

- *each symbol s_j represents an interval range* $s_j := [s_j^l \text{ to } s_j^u)$

- *symbol ranges are disjoint and adjacent* $s_j^u = s_{j+1}^l \; \forall j < k$

- *symbols range over the entire value range* $s_1^l = MINVALUE$ *and* $s_k^u = MAXVALUE$

- *q is a mapping of the values of t to covering symbol ranges* $q_i = s_j$ *iff* $s_j^l \leq t_i < s_j^u$

Quantization is thus a transformation of the continuous value range of the time series to relatively few symbolic representatives of value intervals. Different symbol quantization techniques may be used. The most common ones are equiwidth and equidepth quantization. Equiwidth quantization divides the value range into intervals of equal length. Each of these intervals corresponds to one symbol. Equidepth quantization tries to create intervals with roughly the same number of entries each. SAX (symbolic approximate aggregation) uses the quantiles of the normal distribution [20]. fis r Reconsider our separator example for quantization (Figure 2). Symbol ranges are A: 0-2, B: 2-4, C: 4-6, and so on. Mapping the bottom time series of the left subtree (8.67,8.18,8.72) to these symbols yields (E,E,D). To separate it from the first entry in the next subtree (8.90,10.10,7.85) $\rightarrow$ (E,E,E), more than the one dimension (8.9) is required (see also first example). The separator has to be extended to EEE, thus automatically providing information not only on the first time series dimension, but on three dimensions.

The TS-tree combines quantized separators with additional meta data, i.e. upper and lower bounds of the time series in a subtree:

DEFINITION 4. ***TS-tree inner node.***
An inner node of the TS-tree with branching factor m, rough quantization parameter r and fine quantization parameter f fulfills the following properties:

- *a inner node contains k entries ($m \leq k \leq 2m$) with k pointers to subtrees, $k-1$ separators and k meta data bounds.*

- *a root inner node contains k entries ($2 \leq k \leq 2m$) with k pointers to subtrees, $k-1$ separators and k meta data bounds.*

- *separators are roughly quantized to r symbols and prefix compressed, i.e. common prefixes are not materialized.*

- *meta data upper and lower bounds are finely quantized to f symbols.*

DEFINITION 5. ***TS-tree leaf node.***
A leaf node of the TS-tree with branching factor n and fine quantization parameter f fulfills the following properties:

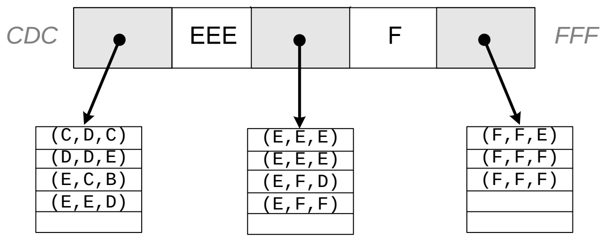


Figure 2: Quantized separators

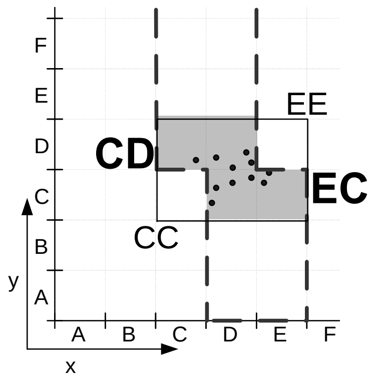


Figure 3: Descriptor

- *a leaf node contains i entries ($n \leq i \leq 2n$) with i pointers to time series.*

- *a root leaf node contains $1 \leq i \leq 2n$ entries.*

- *entries are finely quantized to f symbols.*

Thus, the TS-tree uses rough quantization for long separators on inner nodes while keeping more information through fine quantization of meta data and leaf node entries.

The combined meta data and separator descriptors are illustrated in Figure 3: meta data bounds are lower and upper bounds in each dimension. In this example, dimensions x and y both have lower bounds C and upper bounds E, corresponding geometrically to a (hyper-)rectangle. The separator information, CD to EC however corresponds to the area stretching to the end of the data range in the C interval of dimension x above D in dimension y, the entire data range in the D interval of dimension x and finally from the beginning of the data range in the E interval up to the C interval in dimension y. The intersection of the two geometries, i.e. the indexed subtree range, is marked in gray.

The definition of TS-tree nodes is illustrated in Figure 4: nodes contain both quantized compressed separators as well as meta data information to describe the quantized time series in the subtrees.

Dimensionality reduction of time series is employed in TS-trees to ensure reasonable fanout. Typical time series are long, and dimensionality reduction limits storage requirements for time series. The basic idea is to represent the original time series by a shorter representation, keeping as much information as possible. Different approaches for generation of shorter representations may be used, e.g. PAA (piecewise aggregate approximation) a specialized approach for time series [18]. PAA replaces the original time series

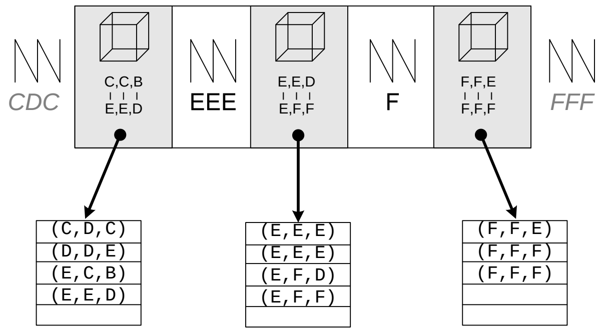


Figure 4: Structure of TS-trees

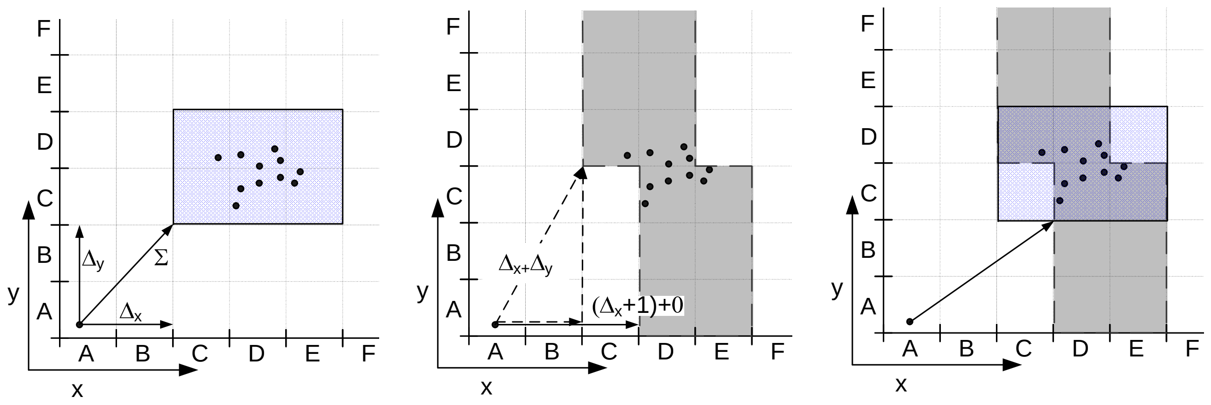


Figure 5: mindist to meta data (left), to separators (center), and to intersection (right)

by piecewise averages of the time series values. The new time series length is thus the old length divided by the length of pieces. Another approach commonly used for time series is DWT (discrete wavelet transform). DWT with Haar wavelet basis builds successive averages and differences to these averages to aggregate the original dimensions. Alternatively, general purpose dimensionality reduction techniques like PCA (principal components analysis) may be used. PCA analyzes the statistical covariance in the dimensions. Using eigenvalue decomposition on the covariance matrix, the original dimensionality may be reduced to the first (w.r.t their variance) of the resulting new dimensions. We analyze the performance of these three methods in the experiments. This concludes the formalization of the TS-tree. In the next section we discuss query processing based on the descriptor information.

5. QUERY PROCESSING IN TS-TREES

Comparing time series is usually done using the Euclidean distance or the Dynamic Time Warping Distance (DTW). While the former simply compares synchronous time series values, DTW allows for stretching and squeezing of the time series in the time dimension. In either case, k-nearest neighbor processing in index structures by appropriate algorithms requires computation of $mindist$: at any node of the tree, the minimal distance between the query and the descriptor information in the node has to be calculated. This is crucial for pruning of subtrees without loss of completeness which are dissimilar and for ranking of potentially relevant nodes in k nearest neighbor search (or for discarding nodes exceeding the range threshold of range queries). We first give the $mindist$ for TS-trees using Euclidean distance. The extension to DTW, which is based on the $mindist$ for Euclidean distance is discussed subsequently.

5.1 TS-tree $mindist$

During query processing in the TS-tree, the query $q = (q_1, \ldots, q_n)$ is compared against the descriptor information $\mathbf{D}$. The descriptor contains both separators and meta data $\mathbf{D} = (\mathbf{S}, \mathbf{MD})$, which both delimit the region indexed by the corresponding subtree. Consequently, the $\mathbf{S}$-$mindist$ to left and right separators $\mathbf{S_l}$ and $\mathbf{S_r}$, as well as the $\mathbf{MD}$-$mindist$ to meta data lower and upper bounds l and u, could be computed. We compute an even tighter overall $mindist$ to the intersection of the two regions as illustrated in Figure 5. The leftmost image depicts the $\mathbf{MD}$-$mindist$ from query q, the center image the $\mathbf{S}$-$mindist$, and the rightmost image the overall $mindist$, respectively. As we can see, the overall $mindist$ is neither the $\mathbf{MD}$-$mindist$ nor the $\mathbf{S}$-$mindist$, but is actually located on the intersection of the regions. Separators for the leftmost and rightmost entry of a node can be obtained from the corresponding parent during query processing. For the leftmost and rightmost entry in the root node we assume $\mathbf{S_l} = (-\infty)$ and $\mathbf{S_r} = (+\infty)$.

The distance to upper and lower boundaries of the meta data can be computed in a straightforward manner as a sum over dimension by dimension differences:

Definition 6. $\mathbf{MD}$-*mindist.*
The $\mathbf{MD}$-*mindist between query* $q = (q_1, \ldots, q_n)$ *and meta data* $\mathbf{MD} = ((l_1, u_1), \ldots, (l_n, u_n))$ *is defined as:*

$$\mathbf{MD}\text{-}mindist(q, \mathbf{MD})^2 = \sum_{i=1}^{n} \begin{cases} (q_i - l_i)^2 & q_i < l_i \\ (q_i - u_i)^2 & u_i < q_i \\ 0 & else \end{cases}$$

The meta data is simply a dimension by dimension information on lower and upper bounds, whereas the separator information is lexicographically ordered, which cannot be assessed in a dimension related manner. Consequently, $mindist$ computation for these two descriptor types differs. Considering Figure 6 (left), we see that the distance from the query to the separator is not addition of differences dimension by dimension. Clearly, lexicographically, the query is smaller than the separator. Computing the distances dimension by dimension, the query would be considered larger than the separator in the second dimension. To compute the $\mathbf{S}$-$mindist$ correctly, we thus have to take previous dimension into account and cannot compute in a dimension wise fashion.

As illustrated in the center image of Figure 5, the $\mathbf{S}$-$mindist$ is either the difference in the current dimension Δ_x plus the distance in the next dimension y or, if the separator in dimension y is larger than the query as is the case in this example, one step further in dimension x, i.e. $\Delta_x + 1$ to immediately reach the separator bounds.

Additionally, we have to take care not to exceed the right separator bounds. The center image of Figure 6 shows that the right separator in the next dimension may be actually smaller than the query. In this case, computing $\Delta_x + 1$ would yield a wrong result of the $mindist$. Summing up all of these cases, we give a recursive definition of the $\mathbf{S}$-$mindist$ where the query is smaller than the separator. The $\mathbf{S}$-$mindist$ from the right is analogous.

Definition 7. $\mathbf{S}$-*mindist.*
The $\mathbf{S}$-*mindist between query* $q = (q_1, \ldots, q_n)$ *and left* $\mathbf{S_l} = (\mathbf{S_{l}}_1, \ldots, \mathbf{S_{l}}_n)$ *and right* $\mathbf{S_r} = (\mathbf{S_{r}}_1, \ldots, \mathbf{S_{r}}_n)$ *separators of a subtree is defined as the distance to the left separator if the query is smaller than the subtree or the distance to the right separator if it is larger. If the query is within the separator bounds,* $\mathbf{S}$-*mindist*

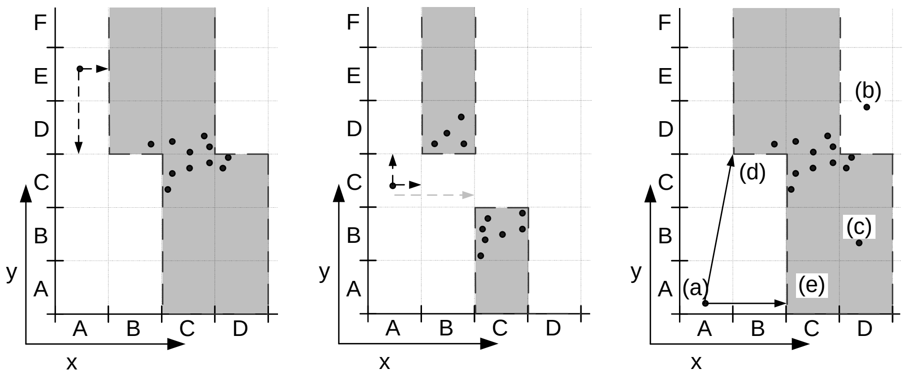


Figure 6: lexicographic separator mindist

is zero:

$$\mathbf{S}\text{-}\mathit{mindist}(q,\mathbf{S}) = \begin{cases} \mathbf{S_l}\text{-}\mathit{mindist}(q_1,\mathbf{S}) & q \le \mathbf{S_l} \quad (a) \\ \mathbf{S_r}\text{-}\mathit{mindist}(q_1,\mathbf{S}) & q \ge \mathbf{S_r} \quad (b) \\ 0 & \mathit{else} \quad (c) \end{cases}$$

where:

$$\mathbf{S_l}\text{-}\mathit{mindist}(q_i,\mathbf{S}) = \min \begin{cases} |q_i - \mathbf{S_l}_i| + \mathbf{S_l}\text{-}\mathit{mindist}(q_{i+1},\mathbf{S}) & (d) \\ |q_i - (\mathbf{S_l}_i + 1)| & (e) \end{cases}$$

$$\mathbf{S_r}\text{-}\mathit{mindist}(q_i,\mathbf{S}) = \min \begin{cases} |q_i - \mathbf{S_r}_i| + \mathbf{S_r}\text{-}\mathit{mindist}(q_{i+1},\mathbf{S}) & (d') \\ |q_i - (\mathbf{S_r}_i - 1)| & (e') \end{cases}$$

Note that in rare cases, the separator ends prematurely (cf. center image of Fig 6) and the minimum is reduced to (d), (d'), i.e. (e) or (e') does not apply iff:

$$\begin{aligned} (e) &: \mathbf{S_l}_i + 1 < \mathbf{S_r}_i \vee (\mathbf{S_l}_i + 1 \le \mathbf{S_r}_i \wedge \mathbf{S_l}_{i+1} \le \mathbf{S_r}_{i+1}) \\ (e') &: \mathbf{S_r}_i + 1 < \mathbf{S_l}_i \vee (\mathbf{S_r}_i + 1 \le \mathbf{S_l}_i \wedge \mathbf{S_r}_{i+1} \le \mathbf{S_l}_{i+1}) \end{aligned}$$

Thus, if the query is smaller than the left separator (a), the $\mathbf{S}$-$\mathit{mindist}$ is the recursively defined left sided distance. Likewise, if the query is larger than the right separator (b), the analogous recursion from the right applies. If the query is inside the region delimited by the separators (c), the $\mathbf{S}$-$\mathit{mindist}$ is clearly zero.

The recursive $\mathbf{S_l}$-$\mathit{mindist}$ takes two aspects into account: for one, as long as the separator bounds have not yet been reached, the difference in the current dimension may contribute to the overall distance and the same choice applies recursively for the next dimension (1). Second, as soon as the bounds are reached by going further an additional symbol, the remaining differences are zero (2). (a) is a recursive choice denoted as $\mathbf{S_l}$-$\mathit{mindist}(q_i,\mathbf{S})$. Moreover, care has to be taken, not to exceed the right separator. Likewise, (b) is the symmetric case of (a) for queries which are larger than the right separator. The $\mathbf{S_l}$-$\mathit{mindist}$ is then the minimum of these choices in reaching the separator boundaries.

Reconsidering Figure 5 (right), we see that the overall $\mathit{mindist}$ is based on both the $\mathbf{MD}$-$\mathit{mindist}$ and the $\mathbf{S}$-$\mathit{mindist}$. In a recursive fashion, as for the $\mathbf{S}$-$\mathit{mindist}$, we proceed until the separator bounds are reached. At this point, however, we may take the $\mathbf{MD}$-$\mathit{mindist}$ for the remaining dimensions into account to exploit the additional meta data information and reach a larger overall $\mathit{mindist}$. Note that a larger $\mathit{mindist}$ is favorable as this allows for better pruning.

DEFINITION 8. ***mindist.***
The mindist between query $q = (q_1, \ldots, q_n)$ and descriptor $\mathbf{D} = (\mathbf{MD}, \mathbf{S})$ of a subtree is defined as:

$$\mathit{mindist}(q,\mathbf{D}) = \begin{cases} \mathit{l\text{-}mindist}(q_1,\mathbf{D}) & q \le \mathbf{S_l} \quad (a) \\ \mathit{r\text{-}mindist}(q_1,\mathbf{D}) & q \ge \mathbf{S_r} \quad (b) \\ 0 & \mathit{else} \quad (c) \end{cases}$$

where:

$$\mathit{l\text{-}mindist}(q_i,\mathbf{D}) = \min \begin{cases} |q_i - \mathbf{S_l}_i| + \mathit{l\text{-}mindist}(q_{i+1},\mathbf{S}) & (d) \\ |q_i - (\mathbf{S_l}_i + 1)| + \mathbf{MD}\text{-}\mathit{mindist}((q_{i+1},\ldots,q_n)) & (e) \end{cases}$$

$$\mathit{r\text{-}mindist}(q_i,\mathbf{D}) = \min \begin{cases} |q_i - \mathbf{S_r}_i| + \mathit{r\text{-}mindist}(q_{i+1},\mathbf{S}) & (d') \\ |q_i - (\mathbf{S_r}_i - 1)| + \mathbf{MD}\text{-}\mathit{mindist}((q_{i+1},\ldots,q_n)) & (e') \end{cases}$$

as before, if the separator ends prematurely only (d) or (d') applies, i.e. (e) or (e') does not apply iff:

$$\begin{aligned} (e) &: \mathbf{S_l}_i + 1 < \mathbf{S_r}_i \vee (\mathbf{S_l}_i + 1 \le \mathbf{S_r}_i \wedge \mathbf{S_l}_{i+1} \le \mathbf{S_r}_{i+1}) \\ (e') &: \mathbf{S_r}_i + 1 < \mathbf{S_l}_i \vee (\mathbf{S_r}_i + 1 \le \mathbf{S_l}_i \wedge \mathbf{S_r}_{i+1} \le \mathbf{S_l}_{i+1}) \end{aligned}$$

Thus the overall $\mathit{mindist}$ to the descriptors in TS-tree nodes is the distance to the intersection of left and right separators and the meta data.

5.2 Extension to DTW

The TS-tree is a flexible time series indexing structure that support for Euclidean distance, as seen before, as well as for Dynamic Time Warping (DTW). DTW allows for stretching and scaling of the time axis to detect slightly shifted or scaled patterns. An example is given in Figure 7: two time series are compared using the Euclidean Distance (left) or DTW (right). Horizontal lines indicate which values are matched by the respective distance functions. Both Euclidean distance and DTW are commonly used for time series similarity search [16]. We briefly review DTW before detailing DTW-based query processing on TS-trees.

DTW computes the best possible match between time series with respect to the overall warping cost. Typically, warping is restricted

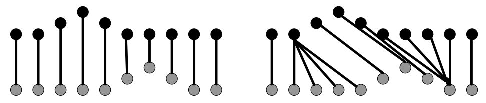


Figure 7: Euclidean Distance (left) and DTW (right)

to some k-band neighborhood around a time series element to avoid degenerated matchings (e.g. all but one values of a time series are matched to a single element of the other). Formally, the definition of local k-band DTW is:

DEFINITION 9. ***k-band DTW.***
The Dynamic Time Warping distance between two time series s, t with respect to a bandwidth k is defined as:

$$D^2_{DTW}(s,t) = D^2_{DTW}(band_k(s), band_k(t)) + \min \begin{cases} D^2_{DTW}(start(s), start(t)) \\ D^2_{DTW}(s, start(t)) \\ D^2_{DTW}(start(s), t) \end{cases}$$

with

$$D^2_{DTW}(band_k(s), band_k(t)) = \begin{cases} D^2_{DTW}(s_i, t_i) & |i-j| \leq k \\ \infty & else \end{cases}$$

Thus, DTW is defined recursively on the minimal cost of possible matches of prefixes shorter by one element. There are three possibilities: either match prefixes of both s and t, or match s with the prefix of t, or vice versa. The difference between overall prefix lengths is restricted to a band of width k in the time dimension by setting the cost of all overstretched matches to infinity. Euclidean distance can be seen as a special case of DTW with a band of width zero. DTW can be computed via a dynamic programming algorithm in $O(mn)$ time and space, where m, n are the lengths of the time series. Using a k-band, this is reduced to $O(k * \max\{m, n\})$. For efficient query processing this is still a limiting factor. Indexing DTW is possible by exploiting k-band restriction. Without this restriction, arbitrary stretching hinders pruning. Consequently, indexing k-DTW using lower bounds has been proposed [16, 30]. An "envelope" around the time series with respect to the k-band is defined. The squared Euclidean distance between values above or below the envelope of the other time series lower bound the actual k-DTW distance. We recall the closest existing lower bound:

$$DTW_{LB} = \sqrt{\sum_{i=1}^{n} \begin{cases} (s_i - U_i)^2 & s_i > U_i \\ (s_i - L_i)^2 & s_i < L_i \\ 0 & else \end{cases}}$$

$$\overline{U}_i = \frac{N}{n}(u_{\frac{n}{N}(i-1)+1} + \cdots + u_{\frac{n}{N}(i)}) \text{ and}$$

$$\overline{L}_i = \frac{N}{n}(l_{\frac{n}{N}(i-1)+1} + \cdots + l_{\frac{n}{N}(i)})$$

A detailed discussion of this bound can be found in [30]. It is an extension of the original envelope for exact indexing of Dynamic Time Warping, LB_{Keogh} [16].

As we can see from the definition of the lower bound, this approach is a dimension-wise computation of distances. Thus, the $mindist$ function introduced in the previous section can be extended to DTW in a straightforward manner by computing the $mindist$ not to a single query time series, but to upper and lower envelope time series.

Lower bounds are useful as filters in multistep query processing. They are used to efficiently compute a set of candidates which is then refined using the original distance function. We review multistep architectures in the next section.

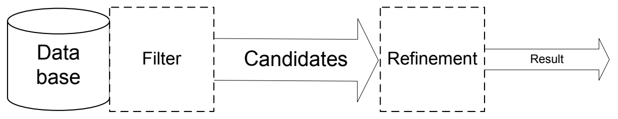


Figure 8: Multistep query procesing

5.3 Multistep query processing

Multistep query processing was introduced in the GEMINI framework (GEneric Multimedia object INdexIng, [13]). In a first step, the set of candidates is generated using a filter distance on the index structure. These are then refined sequentially using the original distance (Fig. 8). To ensure completeness, filter distances must fulfill the lower bounding property. Dimensionality reduction and quantization as proposed above fullfil the lower bounding property for euclidean distances. Shasha et al. proved that reducing the DTW band also fulfills the lower bounding property.

The number of refinement steps is reduced to its minimum in KNOP (k nearest neighbor optimal query processing) [25] by a direct feedback loop between refinement and candidate generation: objects are ordered according to their filter distance via a ranking query and refined until the filter distance is larger than the largest exact distance so far (Fig. 9). Besides reduction of refinement computations, the KNOP approach is especially useful for our evaluation of different time series indexing structures: the number of time series which has to be refined using the original distance function is independent of the indexing structure used. This is due to the ranking query used on the index: since time series are processed in the same order regardless of the index, the number of time series which have to be refined is fixed. Thus, directory page accesses measure the performance of indexes for identical time series representations.

THEOREM 1. ***Number of refinement computations.***
The number of refinement computations in KNOP query processing is constant for any index.

PROOF. Let q denote the query time series, t_i, t_j denote database time series with dimensionality reduced representations r_i, r_j, r_q, respectively. Then, as required in KNOP, filter distances underestimate the exact distance (**lower bounding property**):

$$dist_f(r_i, r_q) \leq dist_e(t_i, q)$$

KNOP uses a ranking query, thus t_j is processed before t_i if its filter distance is smaller (**priority queue**):

$$dist_f(r_i, r_q) \leq dist_f(r_j, r_q)$$

KNOP refines time series until the filter distance of the next time series t_n in the queue exceeds the kth nearest neighbor candidate refined so far: (d_{max} **pruning**):

$$dist_f(r_n) \geq d_{max}$$

with $d_{max} := \max\{dist_e(t_{c_1}), \ldots, dist_e(t_{c_k})\}$. where c_1 through c_k denote the current candidate k nearest neighbors. Thus, time series are only refined until the next reduced representation has a higher filter distance than the exact distance of the k best original representations. In summary, the set of time series refined Ref is

$$Ref := \{t_i | dist_f(t_i) \leq d_{max}\}$$

regardless of the underlying index structure. □

Note that for different representations, e.g. quantizations or dimensionality reductions, the number of refinements may still vary.

Since the number of refinement steps given a certain time series representation cannot be reduced by any indexing structure, performance depends on the number of directory node accesses.

6. EXPERIMENTS

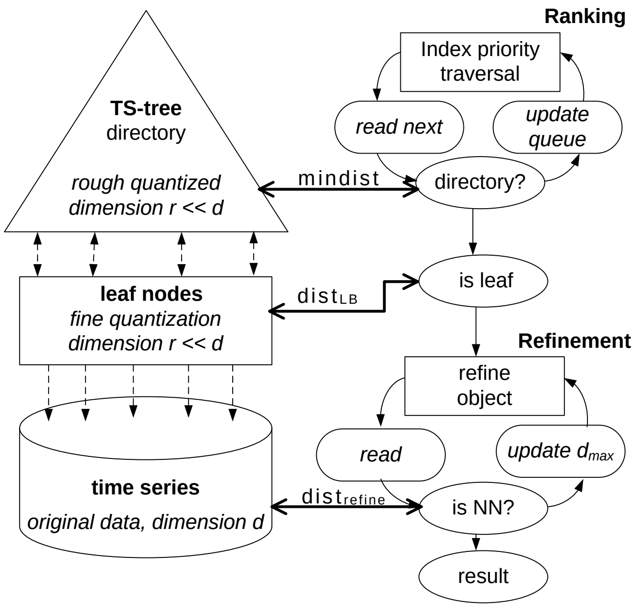


Figure 9: Optimal multistep query processing

Tree	Type	Dimension				
		16	20	24	28	32
R*-Tree	Non-Leaf	5.46	4.60	3.92	3.26	2.48
	Leaf	10.22	8.72	7.61	6.28	5.61
A-Tree	Non-Leaf	5.24	3.85	3.07	2.47	1.69
	Leaf	24.07	18.19	13.81	10.52	7.59
TS-Tree4	Non-Leaf	14,84	12.10	10.96	9.82	8.30
	Leaf	30,47	25.52	22.62	19.13	17.13
TS-Tree16	Non-Leaf	15.23	12.71	11.04	9.82	8.80
	Leaf	32.91	27.64	24.03	20.96	18.68
TS-Tree64	Non-Leaf	15.18	13.27	11.22	9.97	9.12
	Leaf	35.19	29.33	25.12	22.04	19.65

Figure 10: Capacity: average number of entries per node

The experiments are divided into three sections. First, we investigate the structure of TS-tree by measuring the average distances of the node entries and the capacity of the constructed nodes. The next section evaluates the performance of the TS-tree using Euclidean Distance based nearest neighbor queries on synthetic data followed by experiments on real world data sets. The last section evaluates Dynamic Time Warping (DTW) nearest neighbor queries. Hierarchical indexing structures like R-trees or TS-trees grow very slowly in height when indexing more data. Thus very large data sets are used to evaluate indexing structures of appropriate height (four or five).

Data sets.

The experiments presented in this paper use the following data sets (two synthetic and three real world data sets):

1. **RW1**: A synthetic data set based on the random walk model one. The increments between two consecutive values are independent and identically distributed. We use a standard normal distribution $\mathcal{N}(0,1)$ for each increment: $t_{i+1} = \mathcal{N}(t_i, 1)$. We generated four different data sets each containing 250,000 time-series of length 256, 512, 1024 and 2048, respectively.

2. **RW2**: The second version of the random walk model uses independent but not identically distributed values for the increments. The increments of RW2 time series depend on the last increment: $t_{i+1} = \mathcal{N}(2t_i - t_{i-1}, 1)$. We also generated four different data sets each containing 250,000 time-series of length 256, 512, 1024 and 2048, respectively.

3. **Financial**: Historical financial time-series extracted from http://finance.yahoo.com. We extracted end of day stock quotes for more than 8,000 indices, corporate bonds, warrants etc. Overall the data set consists of 1,000,000 overlapping time series of length 256.

4. **Weather**: The weather data set contains 1,000,000 overlapping time series of length 128. The temperature data was retrieved from an interconnection of 127 weather stations gathering agrarian meteorological data.

5. **EEG**: 128Hz-electroencephalographic data from the department of physiology (University of Bologna). Available from the UCR time series archive [17]. We extracted 1,000,000 overlapping EEG time series of length 128.

The synthetic data experiments demonstrate the scalability of the TS-tree while the real world data illustrates the performance of the TS-tree in practical applications. The financial and random walk data set are mean and variance normalized [19]. These data sets are quantized using symbols based on the quantiles of the normal distribution (SAX [20]). The temperature and EEG data set is not normalized and thus indexed using equiwidth symbols. Disk page settings are 1kb and 4kb for synthetic and real world data sets, respectively. The temperature data set is very smooth while the financial data set contains some short changes. The last data set, the EEG data contains a lot of bursts.

Experimental setup.

We compare the TS-tree to the R*-tree [5], an extension of the R-tree [15]. The R*-tree and R-tree both use minimum bounding rectangles (MBR) to describe the data contained in a subtree. We also investigated the performance of the R-tree with linear and quadratic split but as the R*-tree always outperformed the R-tree we only report results of the R*-tree. The A-tree, a recent hierarchical indexing structure which approximates MBRs and data objects by quantized symbols, is also included in the experiments. Each node of the A-tree stores an exact representation of the subtree and quantized VBRs (virtual bounding rectangles) for subtrees. The A-tree originally uses $q = 6$ or $q = 12$ in its experimental evaluation. These quantizations would need bit shifting and masking which is very time consuming. We performed preliminary experiments with A-tree of $q = 8$ and $q = 16$, where the latter showed much worse performance due to lower fanout. We thus set $q = 8$, i.e. quantization with $2^8 = 256$ symbols. The TS-tree is also set to 256 symbols for the leaf nodes. Where less symbols are used by TS-tree separators (this is a parameter we vary in the experiments) we still use one byte to store each symbol for simplicity. The R*-tree implementation uses uses four bytes (floats) per dimension to store the continuous values of the MBRs. All indexing structures (the R*-tree, the A-tree and the TS-tree) are implemented using Java 1.6. We use implementation invariant performance measure like the number of pages accessed or the size of the regions indexed by a subtree as well as the number of refinements necessary.

6.1 Evaluation of structural tree properties

This section studies the indexing structure of TS-trees, R*-trees and A-trees. We measure capacity and compactness of nodes.

Capacity.

The first experiment investigates the average capacity of the index nodes. Figure 10 shows the storage capacity averaged for the data sets RW1 and RW2. Overall 250,000 time series of length 256 are stored by each index and reduced to dimensionality 16 to 32. In this experiment we use PAA to reduce the dimensionality.

Increasing resolution of the separators of TS-tree leads to longer separators, slightly reducing fanout. Consequently, using more symbols leads to better capacity results. Overall, the average capacity of all TS-trees inner nodes is much higher than that of the index structures. This is due to compact symbolic representation. The inner nodes of the A-tree do not show a significantly higher fanout than the nodes of the R*-tree. This is due to the storage overhead for exact MBR and exact centroid per subtree in addition to quantized VBRs. The leaf nodes of the TS-trees also show the highest average storage capacity due to quantization and compression of common prefixes.

Compactness.

The second experiments investigates compactness of subtrees. We measure the average width per indexed region, i.e. levelwise average distances of descriptors. We normalize the tree structures R*-tree, A-tree, and TS-tree, to approximatively the same number of leaf nodes per index in trees of height five. This normalization means that levelwise average widths are comparable across all three indexes. Figure 11 presents the result for the three inner levels of each index for the RW2 data set. “Level 1” denotes all nodes directly above leaves, “Level 2” two levels above leaves and “Level 3” three levels above leaves, respectively. The A-tree shows slightly higher average width than the R*-tree. This is due to the fact that the A-tree stores approximated VBRs, that are actually lower bounds of the exact MBRs. Both A-tree and R*-tree have greater width, which is probably due to overlap of large MBR regions. The compactness of the TS-tree depends on the resolution of the separators. As discussed before, rough quantization means that similar time series have the same quantization and are thus located in the same subtree. As we can see from the graphs, the roughest quantization (only 4 symbols) performs best. It is not only better than resolutions using 16 or 64 symbols, but is clearly more compact than R*-tree or A-tree. Small average width for regions leads to better pruned during query processing, as each subtree is more clearly separated from the others.

Quantization.

As claimed in Section 3.2, splitting of the most informative dimensions yields more compact descriptors. This is illustrated in a more detailed analysis of average width for individual dimensions in Figure 12 for“Level 2” nodes. The separator split in TS-trees splits dimensions successively. Thus, as can be seen in the left part of the figure, the TS-tree has much smaller average width in the first dimensions than in the later ones. In contrast to the TS-tree, both the A-tree’s and the R*-tree’s strategy to globally optimize the MBR split falls behind. Even the average width of the TS-tree4’s last dimension is lower than that of the R*-tree. The right part of Figure 12 analyzes the effect of different separator resolutions for TS-trees. As discussed before, rough quantization indeed leads to longer separators, i.e. later dimensions are split as well. The TS-tree with 4 symbols shows lower average width for later dimensions, and slightly higher average width for the first dimensions. As seen in the previous experiment in Figure 11, TS-tree4 performs best overall. As the TS-tree benefits from using less symbols for separators, we report only TS-trees with 4 or 16 symbols for separators in the following.

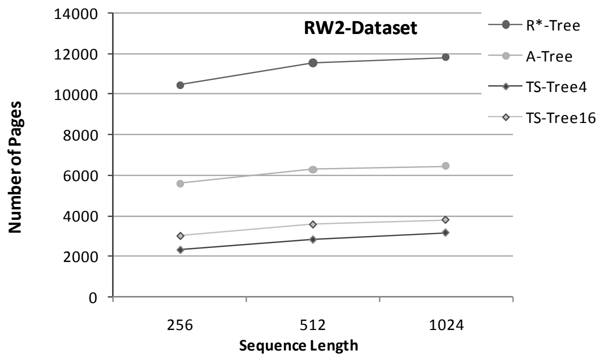


Figure 14: Number of pages accessed for NN queries on RW1 time series of varying length

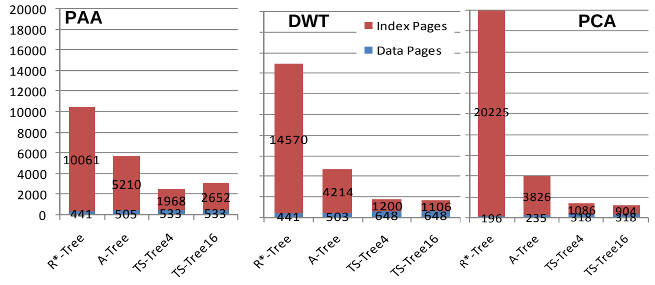


Figure 15: Pages access vs. dimensionality reduction

6.2 L_2 norm time series similarity search

In this section we investigate the scalability and the query performance of the TS-tree on synthetic and real world data sets using the L_2 norm, i.e. Euclidean distance.

Scalability with respect to dimensionality.

To analyze scalability, we report the number of pages read averaged over 25 nearest neighbor queries. For each query we count both index and data pages read. The more the dimensionality of a time series data set is reduced the more time series typically have to be refined in multistep query processing. Thus scalability in terms of dimensionality is very important for indexing time series. Figure 13 illustrates the results for the RW1 and RW2 data set for time series length 256, reduced to dimensionality 16 to 32 using PAA. The TS-tree scales very well for both data sets. This is mainly due to the fact that its overlap-free separator split is not impaired by effects of high dimensional data spaces ("curse of dimensionality"). The R*-tree and A-tree, however, fall prey to the curse of dimensionality and degrade with increasing dimensionality (cf. also [5, 7, 6]). For the RW2 data set the TS-tree is nearly not influenced by the dimensionality of the data. This is due to decreasing number of refinement steps. For 16 dimensions 8% of the pages read are refinement steps while only 4% are refinement steps if 32 dimensions are indexed. Thus, more dimensions require less refinement page reads whereas fewer dimensions require less directory page reads.

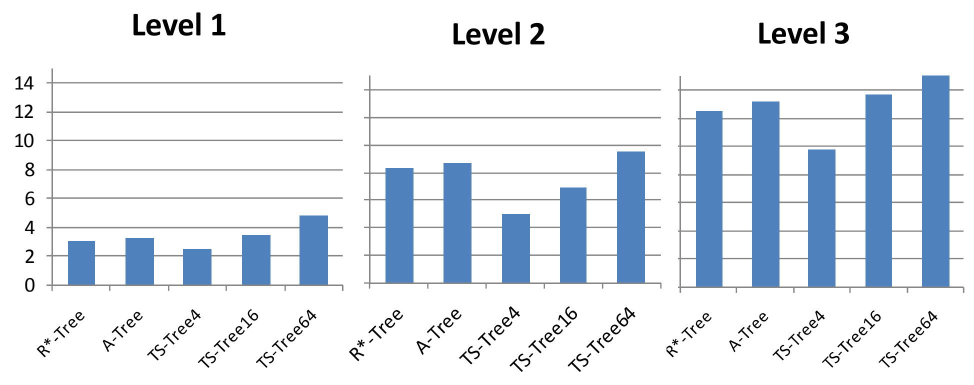


Figure 11: Compactness: average width per level

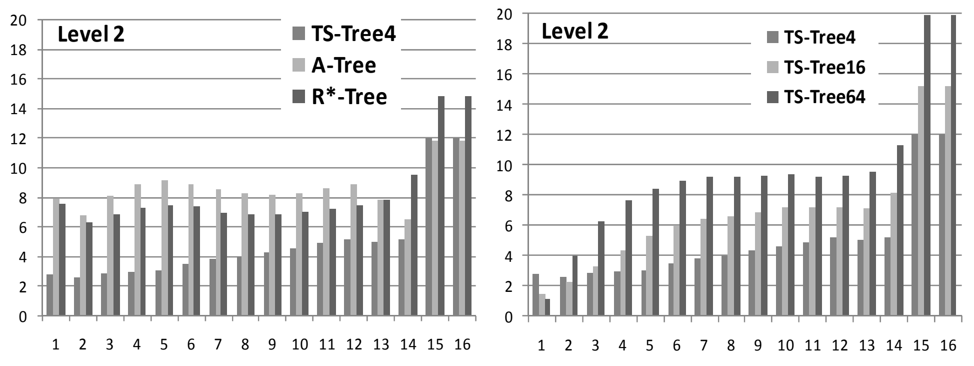


Figure 12: Compactness: average width per dimension

Overall, the TS-tree outperforms both A-tree and R*-tree by nearly one order of magnitude.

Scalability with respect to time series length.

We generated time series of different length (256, 512, 1024) and reduced them to 16 dimensions. As before, the TS-tree outperforms the R*-tree and A-tree (see Figure 14). With increasing length the number of refinement steps increases which influences all index structures in the same way, as proven in Section 5.3. As shown in the previous experiment, TS-trees may compensate this effect by indexing more dimensions.

Performance of dimensionality reduction techniques.

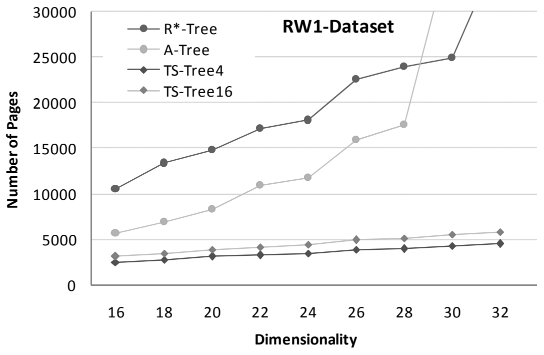
We investigate three dimensionality reduction techniques: piecewise aggregate approximation (PAA), discrete wavelet transform with Haar basis (DWT) and principal component analysis (PCA). As mentioned before, PCA orders new (reduced) dimensions in descending order of their variance. DWT orders the dimensions according to their level of detail. Since TS-trees split the dimensions according to their order, it benefits from those dimensionality reduction techniques that provide an inherent ordering. Figure 15 shows the number of pages read, depicting data pages in blue (lower bar), and index pages in red (upper bar) for the RW1 data set. As we can see, the R*-tree fails to capture the inherent dimensionality order. While the A-tree and TS-tree benefit from the ordering, the R*-tree actually degrades. Note that the number of refinement steps is identical in DWT and PAA as both reduce to the same details in time series. However, the ordering is different, which results in different splits during index construction. Although the TS-tree accesses more pages for refinement than the other two indexes, it always performs more than twice as good as the A-tree, due to much more compact index pages. Using 16 instead of 4 symbols further reduces page access in TS-trees using DWT or PCA. One reason for this is rough quantization. Using only four symbols, the first dimensions are split less often. As these first dimensions carry more information in PCA or DWT, using more symbols makes better use of the inherent ordering by focusing split to the first dimensions. Thus, we use 16 symbols for the separators in the following.

Performance on real world data.

We investigate the performance of the TS-tree on three real world data sets.

Performance on smooth time series.

The left part of Figure 16 shows the result for the financial data reduced to 16 dimensions using DWT. As DTW shows much better results than PAA we only report the result for DWT. Like before the R*-tree reads a huge amount of index pages. Again the TS-tree needs more refinement steps than the A-tree but only half the number of index pages (the A-tree needs 3,552 and the TS-tree 1,785). The right part illustrates results for the time series weather data. Since the weather time series are very smooth, PAA is sufficient to reduce the dimensionality. Dimensionality reduction techniques like PAA capture the main characteristics of smooth time series very well. Hence, for the weather data set only a few refine-



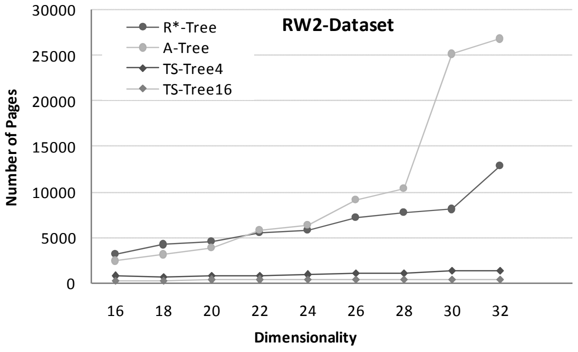
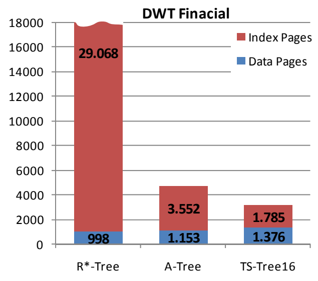


Figure 13: Number of pages accessed for 1-NN queries using RW1 and RW2 time series



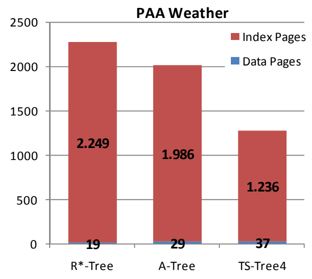


Figure 16: Page access for financial and weather data

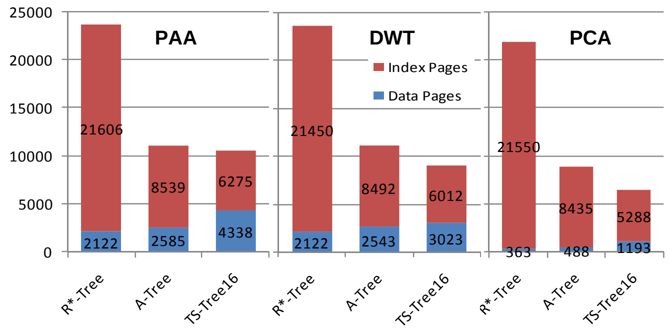


Figure 17: Number of pages accessed for the EEG data for different dimensionality reduction techniques

ment steps are necessary. The TS-tree again clearly outperforms the other two index structures.

Performance on bursty time series.
Next, we analyze the query performance on bursty time series in the EEG data set. Bursts are typically smoothed away by dimensionality reduction techniques and hence more refinement steps are necessary for bursty data. As discussed, the number of refinement steps can be reduced by increasing the number of indexed dimensions. For the EEG data we obtained the best result if 32 dimensions are indexed. Figure 17 shows the result for the number of refinement and index pages read for all three dimensionality reduction techniques. As already evaluated in the last section, the R*-tree slows down as more dimensions are indexed. For the EEG data PCA is able to reduce the number of refinement steps by more than a factor of two. Considering the number of index pages read, the TS-tree also shows better results than the other index structures. Especially for the PCA reduced data set, the TS-tree is again able to reduce the number of index pages read.

6.3 DTW time series similarity search

In this section we evaluate the applicability of DTW queries using hierarchical index structures. In [30] many linear transformations have been tested but PAA has been shown to be the best dimensionality reduction technique DTW queries. Thus we use PAA to reduce the dimensionality of the data with the lower bound presented in Section 5.3.

The first experiment evaluates the influence of different DTW warping width on the query performance (Figure 18). For this experiment we use the RW1 data set. As the increments of the time series are independent using RW1 many time series have to be refined if the warping width is increased. Hence the total number disk pages read quickly increases with increasing warping width. As the number of refinement steps are dominating the number of index pages read, the advantage of using the TS-tree disappears if a higher warping width is used. For a small warping path the TS-tree clearly outperforms the R*-tree and A-tree and even with a higher warping width the TS-tree is still more efficient the the other two index structures.

The last experiment investigates the query performance of DTW time series queries on real world data. We used a warping width of 3 and 5 for the weather and financial data set respectively. For the weather data set, 32 dimensions are indexed and for the financial data set, 64 dimensions are necessary to reduce the number of refinement steps to a reasonable amount. Figure 19 illustrates the result for both data sets. Again, the TS-tree works faster for both data sets. The TS-tree clearly outperforms both other index structures using DTW queries, especially for the weather data set.

7. CONCLUSION

In this work, we demonstrate the importance of overlap-free tight bounding regions in hierarchical indexing structures for efficient time series query processing. We propose the TS-tree which exploits the inherent properties of time series for compact descriptors and overlap-free indexing. Descriptors provide superior pruning power through quantization of separators. Quantization leads to automatic aggregation of similar time series, longer separators and better pruning power. Our $mindist$ takes both separator and meta data into account to prune many index nodes as demonstrated in the

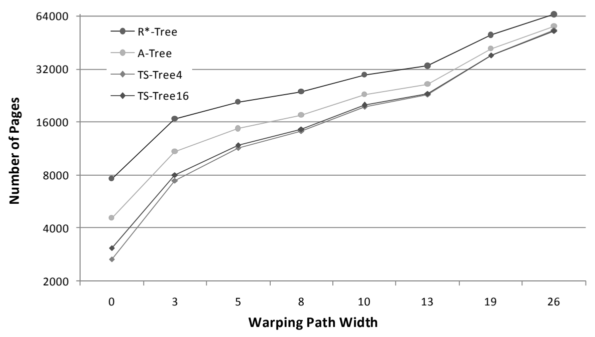


Figure 18: Number of pages accessed for a RW1 data set using different DTW Warping Width

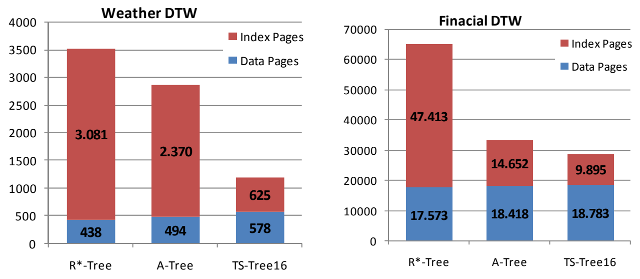


Figure 19: Number of pages accessed for the Weather and Financial data using DTW

experiments. The TS-tree clearly outperforms existing tree structures like the R*-tree and the quantization based A-tree.

8. REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient Similarity Search In Sequence Databases. In *Proc. FODO*, pages 69–84, 1993.

[2] R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indexes. In *ACM SIGFIDET Workshop*, pages 107–141, 1970.

[3] R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indices. *Acta Inform.*, 1:173–189, 1972.

[4] R. Bayer and K. Unterauer. Prefix B-trees. *ACM TODS*, 2(1):11–26, 1977.

[5] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proc. SIGMOD*, pages 322–331, 1990.

[6] S. Berchtold, C. Böhm, H.-P. Kriegel, J. Sander, and H. Jagadish. Independent quantization: An index compression technique for high-dimensional data spaces. In *Proc. ICDE*, pages 577–588, 2000.

[7] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proc. VLDB*, pages 28–39, 1996.

[8] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *AAAI KDD Workshop*, pages 229–248, 1994.

[9] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbors meaningful. In *Proc. ICDT*, pages 217–235, 1999.

[10] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM CSUR*, 33(3):322–373, 2001.

[11] C. Chatfield. *The Analysis of Time Series: an Introduction*. Chapman and Hall, 1996.

[12] S. Chien and N. Immorlica. Semantic similarity between search engine queries using temporal correlation. In *Proc. WWW*, pages 2–11, 2005.

[13] C. Faloutsos. *Searching Multimedia Databases by Content*. Kluwer, 1996.

[14] P. Ferragina and R. Grossi. The string b-tree: A new data structure for string search in external memory and its applications. *J. ACM*, 46(2):236–280, 1999.

[15] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. SIGMOD*, pages 47–57, 1984.

[16] E. Keogh. Exact indexing of dynamic time warping. In *Proc. VLDB*, 2002.

[17] E. Keogh. UCR time series archive, http://www.cs.ucr.edu/˜ eamonn/TSDMA/datasets.html, 2006.

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

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

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

[21] K.-I. Lin, H. V. Jagadish, and C. Faloutsos. The tv-tree: An index structure for high-dimensional data. *VLDB Journal*, 3(4):517–542, 1994.

[22] B. Liu, R. Jones, and K. L. Klinkner. Measuring the meaning in time series clustering of text search queries. In *Proc. CIKM*, pages 836–837, 2006.

[23] H. Sakoe and S. Chiba. A dynamic programming approach to continuous speech recognition. In *Proc. ICA*, pages 65 –68, 1971.

[24] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The A-tree: An index structure for high-dimensional spaces using relative approximation. In *Proc. VLDB*, pages 516–526, 2000.

[25] T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. In *Proc. SIGMOD*, pages 154–165, 1998.

[26] D. Shasha and Y. Zhu. *High performance discovery in time series*. Springer, New York, 2004.

[27] M. Vlachos, C. Meek, Z. Vagena, and D. Gunopulos. Identifying similarities, periodicities and bursts for online search queries. In *Proc. SIGMOD*, pages 131–142, 2004.

[28] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. VLDB*, pages 194–205, 1998.

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

[30] Y. Zhu and D. Shasha. Warping indexes with envelope transforms for query by humming. In *Proc. SIGMOD*, pages 181–192, 2003.