To appear in Proceedings of 12th International Conference on Scientific and Statistical Database Management, Berlin, Germany.

IEEE Computer Society Press

TSA-tree: A Wavelet-Based Approach to Improve the Efficiency of Multi-Level Surprise and Trend Queries on Time-Series Data *

Cyrus Shahabi, Xiaoming Tian, Wugang Zhao
Integrated Media Systems Center & Department of Computer Science
University of Southern California, Los Angeles, California 90089
[shahabi, xtian, wugangzh]@usc.edu

Abstract

We introduce a novel wavelet-based tree structure, termed TSA-tree, which improves the efficiency of multi-level trend and surprise queries on time sequence data. With the explosion of scientific observation data (some conceptualized as time-sequences), we are facing the challenge of efficiently storing, retrieving and analyzing this data. Frequent queries on this data set is to find trends (e.g., global warming) or surprises (e.g., undersea volcano eruption) within the original time-series. The challenge, however, is that these trend and surprise queries are needed at different levels of abstractions (e.g., within the last week, last month, last year or last decade). To support these multi-level trend and surprise queries, sometimes huge subset of raw data needs to be retrieved and processed. To expedite this process, we utilize our TSA-tree. Each node of TSA-tree contains pre-computed trends and surprises at different levels. Wavelet transform is used recursively to construct TSA nodes. As a result, each node of TSA tree is readily available for visualization of trends and surprises. In addition, the size of each node is significantly smaller than that of the original time series, resulting in faster I/O operations. However, a limitation of TSA-tree is that its size is larger than the original time series. To address this shortcoming, first we prove that the storage space required to store the optimal subtree of TSA-tree (OTSA-tree) is no more than that required to store the original time-series without losing any information. Next, we propose two alternative techniques to reduce the size of OTSA-tree even further, while maintaining an acceptable query precision as compared to querying the original time sequences. Utilizing real and synthetic time-sequence databases, we compare our techniques with some well known algorithms such as DFT and SVD in both performance and query precision. The results indicate the superiority of our approach. Finally, we show that our techniques are scalable as we increase either the database size or the length of time sequences.

1. Introduction

With the explosion of scientific observation data (some of which are time-series), we face the challenge of storage, retrieval and analysis of this data. For example, as part of a joint project that we defined with Jet Propulsion Laboratory (JPL) for NASA[1], the temperature, pressure and refractivity of certain coordinates on earth are recorded every hour at different ground stations. This data is going to be populated in databases for on-line access by experts around the world. The data can be represented as a matrix of size $M \times N$, where M is the number of ground stations while N is the number of observations (e.g., temperature) recorded every hour for the past decade. Hence, with these applications, M is in the order of hundreds while N might be in the order of hundreds of thousands. Other types of similar applications are stock quotes of companies for the past 10 years, where $N = 3650$ (assuming only one recording per day), while usually $M = 100$ (fortune 100 companies) or $M = 500$ (fortune 500 companies).

*This research has been funded in part by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152, by NASA/JPL contract nr. 961518, and unrestricted cash/equipment gifts from Intel and NCR.

[1] The project is entitled GENESIS: GPS environmental and earth science information system. In this project, signals from GPS satellites are processed and analyzed to extract the atmospheric data (see http://genesis.jpl.nasa.gov).

Typical queries on the above mentioned data sets are, however, different than conventional *point* queries. For example, a query to acquire the temperature of a specific station at a specific time and date is not very frequent. The more frequent queries are of the nature of identifying (or visualizing) the trends and surprises in the data set at multiple *levels* of abstractions[2]. For example, consider the following queries:

Q1. Find the cities where temperature has been increasing during the last month.

Q2. Find the cities where temperature has been decreasing during the last decade.

These queries look for rising and falling trends. However, it is not usually required that the temperature really go up (or down) at every moment; some "small" movement at the opposite direction is allowed, which is just "noise" comparing to the main trend. The tolerance range of "noise" depends on the length of trends the user is interested in. With Q1, some small downward movement within a day can probably be ignored, while with Q2 some small upward movement within a month can probably be safely disregarded. In other words, some relatively small scale deviation from the general trend can be tolerated when one is looking for an overall large-scale trend. We denote these types of queries as *multi-level trend* queries.

Similarly, frequent queries are submitted to identify surprises within a data set. For example:

Q3. Find the cities where temperature has sudden changes (daily) during the the month.

Q4. Find the cities where temperature has sudden changes (monthly) during the last decade.

Again with Q4, sudden changes within a day or two can be ignored. Furthermore, at the level of years, we may be interested in surprises such as "rising quickly at the first month and decreasing quickly during the next month". We denote these types of queries as *multi-level surprise* queries.

Although at first glance it seems that we need to retrieve and process a huge subset of raw time-series data to support multi-level trend and surprise queries (e.g., 10 years of data needs to be retrieved and processed for Q4), we show that by storing pre-computed trends and surprises at different levels of abstractions, these queries can be supported much more efficiently.

In this paper, we propose a novel tree-like data structure termed *TSA-tree* (stands for Trend and Surprise Abstractions). The root of this tree is the original time series while each internal node (or leaf) is constructed by applying wavelet transform to its parent. We prove that by utilizing the wavelet transform, we can naturally split a time-series sequence into two nodes where one node captures the trends and the other the surprises within the original sequence. Hence, as we traverse down the tree, and applying wavelet recursively to trend sequences, we increase the level of abstraction on trends and surprises. Meanwhile, we show that as we traverse down the tree, the size of nodes decrease by half. Therefore, the higher the level of abstraction required by the trend and surprise queries, the better the performance of the system to support these queries (the rate of improvement is exponential). In sum, the nodes of TSA-trees can immediately be used to visualize trends and surprises at different levels. They not only need a small post processing, but also are much smaller in size as compared to original time series. Hence, the performance is improved due to both eliminating the CPU-bound processing and significantly reducing the I/O cost of retrieval. When a trend/surprise query is submitted, we find the best matched level in TSA-tree, and fetch the corresponding node to display it to the user (for visualization).

In addition, we consider different scenarios where TSA-tree cannot be stored on magnetic disk(s) in its entirety. We prove a specific subset of the tree (specifically, all its leaf nodes) is the optimal subset to be kept on disk, termed OTSA-tree. The size of OTSA-tree is shown to be identical to the size of the original time-series. For the cases where the storage space is scarce, we propose alternative techniques to reduce the size of OTSA-tree further by dropping nodes and/or coefficients with less energy. The resulting condensed OTSA-tree, hence, becomes a competitor to other techniques discussed in the literature such as Discrete Fourier Transform (DFT) [2, 3] and Single Value Decomposition (SVD) [20]. Therefore, we conducted comprehensive experiments to compare our results with those techniques. Briefly, we outperformed SVD and DFT in both performance and accuracy for our application. This is because SVD performs poorly when $N >> M$ and DFT cannot maintain the surprises due to only keeping the first few coefficients.

The remainder of this paper is organized as follows. Section 2 provides some background material on related work. Section 3 introduces TSA-tree and explains the procedure of trends and surprises mining using wavelet transform. In Section 4, we discuss issues in reducing the size of TSA-tree. Section 5 provides a discussion and analysis on our technique and compares it with other traditional methods. Finally, Section 6 concludes the paper and provides an overview on our future work.

[2] We focus our attention on the efficient storage and retrieval of time series data. A good body of work exists on how to process similarity matches on this data that can be used as a complement to our work (see Section 2).

2. Related work

Traditional methods of time series analysis are mainly concerned with decomposing a series into a trend, a seasonal variation, and other "irregular" fluctuations [8, 14].

$$Y_t = T_t + S_t + Z_t \tag{1}$$

where T_t is the "trend" component, S_t is the "seasonal" component, and Z_t is the "irregular" or "random" component. In this model, "trend" is loosely defined as "long term change in the mean". A difficulty with this definition is deciding what is meant by "long term". For example, climatic time series sometimes exhibit cyclic variations over a very long time-period such as a year (four seasons). If one just had one season data, this long time oscillation would appear to be a trend; but if several hundred years data were available, the long term oscillation would be visible. A traditional method of dealing with non-seasonal data which contains a trend is to fit a simple function such as polynomial or logarithmic curve. For all curves of this type, the fitted function provides a measure of the trend. A second procedure for dealing with a trend is to use a *linear filter* which converts one time series $X = (x_1, x_2, ..., x_n)$ into another $Y = (y_1, y_2, ...y_n)$ by the linear operation :

$$y_t = \sum_{i=-m}^{k} a_i x_{t+i} \quad t = 1, 2, ...n \tag{2}$$

where $\{a_i\}$ is a set of weights. In order to smooth out local fluctuations and estimate the local mean, we should clearly choose the weights so that $\sum_i a_i = 1$ and then the operation is often referred to as a *moving average*. *Moving average* is discussed in detail by Kendall [17], and it is widely used in stock data analysis (for example, see [27]). However, model (1) requires pre-defined fitting functions for T_t and S_t , also assumes Z_t is a stationary time series[3], which in some practical cases (sharp changes) is infeasible, and models (1) and (2) cannot handle multi-level trends, i.e., how to deal with "short term" trends as well as "long term" trends.

Database community typically uses a curve-fitting method to obtain trends and movement. In [21, 28] straight lines are used to approximate the time series, and it employs divide-and-conquer technique to segment before approximation, which is time-consuming and loses surprise information. Qu et. al, [24] use line-fitting to obtain multi-scales movement. In their study, a pattern is defined as a regular expression of letters, where each letter describes a movement direction and covers a specified length of time (called pattern unit length). To find that the time series (or part of it) matches a pattern, the time series is first partitioned into consecutive sub-series if the distance between the best fitting line and the sub-series is within a specified tolerance. The complexity of line-fitting is of $O(n^2)$, where n is the length of the input time series. With our method, the complexity can be reduced to $O(n)$ to obtain multi-level trends, and the accuracy is higher due to employment of curve-fitting (via wavelet) versus line-fitting. Finally, to the best of our knowledge, using uniform wavelet-based model to capture both surprises and trends at multi-levels is unique.

Traditional surprise detection methods are usually referred to as $outliers$, or $surprises$ in our daily life. There are two steps in traditional outlier analysis, the first step is generally an inspection of plots of the original data. Such plots help one defines the nature of the data. The plot of the data against time is often sufficient to identify very extreme observations, i.e. $outlier$ [1, 18, 19, 4]. However the outlier detection techniques are expensive when compared with the wavelet transform and it cannot capture multi-level surprises.

In this paper, we also try to provide an efficient way for placing TSA-tree on disk. By reducing data volume, we can achieve less disk I/O and efficiency for some aggregate queries. In [20], Korn et. al use singular value decomposition (SVD) transform to map a huge matrix into a much smaller one. By data compression, they can support ad hoc queries on large volume of data sets. However, there are some problems inherent in the SVD method. With SVD, any $M \times N$ matrix X can be written as the product of a $M \times r$ column-orthogonal matrix U, a $r \times r$ diagonal matrix Λ and a $r \times N$ row-orthogonal matrix V, i.e., $X = U\Lambda V$ (where r is the rank of the matrix X). Subsequently, SVD reconstructs each row of X by multiplying one row of U to the entire Λ and V, thus, the complexity of reconstructing each row is $O(|V|)$, where $|V|$ is the size of V. There are three assumptions in [20] that do not hold within our applications:

1. $M >> N$.
   Within our applications M and N may be in the same order or even $M << N$. Therefore, $O(|V|)$, the complexity to reconstruct a row of X, is not small anymore. To address this problem, if we naively try to apply SVD to the transpose of X (as X'), then SVD will not be optimal anymore due to correlation within the columns of X' and not within its rows. That is, let $Y = X'$. The rows of X become the columns of Y, and we apply SVD such that $Y = X' = V'\Lambda U'$. When we want to retrieve a column of Y, we need the entire V', Λ, and one column of U'. Thus, the complexity is still $O(|V|)$.

2. N is in the order of hundreds.
   However, with our applications, N can be large and hence the complexity of reconstructing one row of X i.e., $O(|V|)$ is large.

3. X is not updated.

[3] For the definition of stationary time series, consult [14].

However, with our applications, X must be updated daily or monthly. SVD has to recompute the whole matrix again to update, but wavelet only need to add new coefficients at the end of each TSA-tree node.

In the database literature, much work has been done on querying time series data. Most of the effort, however, has been on similarity queries [2, 3, 6, 13, 26, 25, 28]. Some of these works can be considered as a complement to our study, to perform similarity match on trends and surprises (i.e., TSA-tree nodes). With data mining applications, it is often necessary to search within a series database for those series that are similar to a given query series in the presence of noise. Agrawal et. al, [2] map time series into the frequency domain using the Fourier transform and keep the first few coefficients in the index for similarity searching, meanwhile, a query processing algorithm that uses the underlying R-tree index of a multidimensional data set is provided to answer similarity queries efficiently. Chan et al. [7] uses wavelet transform instead of the Fourier transform and also keeps the first few coefficients for similarity searching. The shortcoming of these methods is that they drop the surprises.

3. A Tree for Multi-level Trend and Surprise Abstractions (TSA-tree)

In this section, we explain the design of a new wavelet-based data structure, termed TSA-tree, which can be constructed for each time sequence data (i.e., each row of the $M \times N$ matrix). Therefore, we need to keep M TSA-trees per matrix. Later in Section 4, we propose techniques to reduce the size of each row (or TSA-tree) to make it smaller than the original time series. During our performance evaluation, Section 5, we compare the storage requirement of the entire matrix to that required by other techniques.

Wavelet theory involves representing general functions in terms of simpler, fixed building blocks at different scales and positions. This has been found to be very useful in several different areas, such as sub-band filtering, quadratic mirror filters, and pyramid schemes in the area of signal and image processing, for the collections of references see [5, 9, 10, 12, 23]. We can use wavelet transform to obtain multi-level trends and surprises in a uniform scheme. Towards this end, we introduce an operation termed $split$ to generate a multi-level tree, where each node denotes the wavelet coefficients of the corresponding multi-level trends and surprises. In this section we first formally define the split operation. Next, we define an inverse operation, termed $merge$. Subsequently, we prove that both $split$ and $merge$ operations are lossless and reversible. Finally, we show how these two operations are used to build a TSA-tree and how we can utilize TSA-tree for trends and surprises mining.

3.1 $Split$ and $merge$ operations

For time series $X = (x_0, x_1, \ldots, x_{n-1})$, we use$|X|$ to denote the length of X, i.e, the number of data items in X, in this case $|X| = n$. We also use $\|X\|$ to denote the $norm$ of X, i.e., $\|X\| = \sqrt{x_0^2 + x_1^2 + \cdots + x_{n-1}^2}$.

Definition 3.1: $Convolution$ (cyclic) is an operation between two sequences $X = (x_0, x_1, \cdots, x_{n-1})$ and $H = (h_0, h_1, \cdots, h_{l-1})$, the result is a sequence Y with same length as X, where $y_n = \sum_{i=0}^{l-1} h_i x_{n-i}$ when indices are out of range, we pad the sequence with zero values. We denote convolution as $Y = Conv(X, H)$ ∎

Example 3.2: If $X = (2, 1, 3, -1)$ and $H = (0.1, 0.5)$, then $Conv(X, H) = (-0.3, 1.1, 0.8, 1.4)$ ∎

Definition 3.3: $Down - Sampling$ (by 2) is an operation which takes a sequence $X = (x_0, x_1, ..., x_{2n-1})$ as input and produces a sequence $Y = (y_0, y_1, ..., y_{n-1})$ as output, where $y_i = x_{2i+1} \quad for \quad i = 0, 1..., n-1$. Y has half length of X, denoted as $Y = Down - Sample(X)$ ∎

Example 3.4 : For $X = (2, 1, 3, -1)$, $Down - Sample(X) = (1, -1)$. ∎

For a given wavelet transform, two pairs of sequences are associated with it. The first pair is called *wavelet analysis filters* , denoted as H_a, G_a. The other pair is called *wavelet synthetic filters* , denoted as H_s, G_s. They are uniquely determined by the wavelet transform. For example, for the Haar wavelet, the simplest and most popular wavalet given by Haar [15], the wavelet analysis filters associated with Haar wavelet are :

$$H_a = (1/\sqrt{2}, 1/\sqrt{2}) \quad G_a = (-1/\sqrt{2}, 1/\sqrt{2})$$

The wavelet synthetic filters associated with Haar wavelet are :

$$H_s = (1/\sqrt{2}, 1/\sqrt{2}) \quad G_s = (1/\sqrt{2}, -1/\sqrt{2})$$

Now, the $split$ operation can be defined using the $Down - Sampling$ and $Convolution$ operations. That is, for a time sequence $X = (x_0, x_1, ..., x_{n-1})$, we can use $split$ operation to get trend sequence AX and surprise sequence DX. The algorithm for split operation is illustrated in Figure 1. The complexity of split operation is $O(nl)$, where l is the length of filter H and G, and n is the length of original time series.

Reversibly, for two equi-sized sequences AX and DX, $merge$ operation is to obtain a time sequence X by AX and DX, where $|X|$ = 2$|DX|$ = 2$|AX|$. The $merge$ operation can be defined using the $Up - Sampling$ and $Convolution$ operations.

```
{AX, DX} = split(X)
  begin
    Temp-AX = Conv(X, H_a)
    Temp-DX = Conv(X, G_a)
    AX = Down-Sample(Temp-AX)
    DX = Down-Sample(Temp-DX)
  end
```

(a) Algorithm for $split$ operation

```
X = merge(AX, DX)
  begin
    Temp-AX = Up-Sample(AX)
    Temp-DX = Up-Sample(DX)
    X = Conv(Temp-AX, H_s) +
          Conv(Temp-DX, G_s)
  end
```

(b) Algorithm for $merge$ operation

Figure 1. Split and merge operations

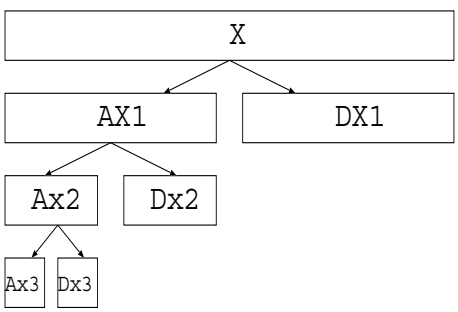


Figure 2. TSA tree

Definition 3.5: $Up-Sampling$ is an operation which takes a sequence Y as input and produces an output sequence X of property: $x_{2n} = y_n$ and $x_{2n+1} = 0$, where X has twice the length of Y, denoted as $Y = Up-Sample(X)$ ∎

Example 3.6: For $Y = (2, 3)$, $Up-sample(Y) = (2, 0, 3, 0)$ ∎

The algorithm for $merge$ operation is illustrated in Figure1. The complexity of $merge$ is also $O(nl)$.

3.2 Properties of TSA-tree

A TSA tree is constructed by applying the split operation on AX_i's repeatedly. That is, we start by applying $split$ on X to obtain AX_1 and DX_1. Subsequently, we split AX_1 into AX_2 and DX_2. This procedure repeats k times in order to construct a TSA tree with $k+1$ levels (see Figure 2).

There are two major properties of TSA-tree that make the structure appropriate for efficient multi-level trends and surprises mining. First, each node of TSA-tree can be reconstructed by "merging" its children without losing any information (i.e., merge is lossless, see Lemma. 3.7). Second, each pair of sibling nodes of TSA-tree preserves the energy of its parent's time series. Below we prove these properties for the Haar wavelet [Mal89].

Lemma 3.7: *We can reconstruct AX_i (Note that $AX_0 = X$) from AX_{i+1} and DX_{i+1} by $merge$ operation, i.e., if $\{AX_{i+1}, DX_{i+1}\} = split(AX_i)$, then, $AX_i = merge(AX_{i+1}, DX_{i+1})$.*

Proof: For[4] $AX_i = (x_0, x_1, x_2, x_3)$, due to the $split$ operation, after $Conv$, we have $Temp_AX_{i+1} = AX_i\mathbf{H}$, $Temp_DX_{i+1} = AX_i\mathbf{G}$, where $\mathbf{H}$ and $\mathbf{G}$ are two matrices determined by *wavelets analysis filter*, for Haar wavelet,

$$\mathbf{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{G} = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 1 & 0 & 0 & -1 \end{pmatrix}$$

After $Down-Sample$,

$$AX_{i+1} = (a_0, a_1) = (x_0, x_1, x_2, x_3)\mathbf{P} \tag{3}$$

$$DX_{i+1} = (d_0, d_1) = (x_0, x_1, x_2, x_3)\mathbf{Q} \tag{4}$$

where

$$\mathbf{P} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

[4] For the sake of illustration, we assumed $|AX_i| = 4$. The extension to larger sizes of AX_i is straightforward.

$$\mathbf{Q} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}$$

Due to the $merge$ operation, after $Up - Sample$ and $Conv$ operations, we have :

$$TX_{i+1} = (x_0, x_1, x_2, x_3) \frac{1}{\sqrt{2}} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{H}$$

$$SX_{i+1} = (x_0, x_1, x_2, x_3) \frac{1}{\sqrt{2}} \begin{pmatrix} 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} (-\mathbf{G})$$

$$TX_{i+1} + SX_{i+1} = (x_0, x_1, x_2, x_3) = AX_i$$

∎

Lemma 3.8: *$Split$ is an inverse operation of $merge$, i.e., if $AX_i = merge(AX_{i+1}, DX_{i+1})$, then $\{AX_{i+1}, DX_{i+1}\} = split(AX_i)$.*

Proof: This can be shown by multiplying the matrix $\mathbf{P}$ to both sides of Eq.(5) to obtain $AX_{i+1} = AX_i\mathbf{P}$. Similarly, by multiplying the matrix $\mathbf{Q}$ to both sides, we obtain $DX_{i+1} = AX_i\mathbf{Q}$. ∎

In addition, we prove that $split$ preserves the energy of its original time series. That is, there is no false dismissal with the $split$ operation.

Lemma 3.9: $\|X\|^2 = \|AX_1\|^2 + \|DX_1\|^2$.

Proof: For $X = (x_0, x_1, x_2, x_3)$, given Eq. (3)(4) and combining $\mathbf{P}$ and $\mathbf{Q}$ into one matrix, we obtain :

$$\mathbf{C} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix}$$

Note that matrix $\mathbf{C}$ is an orthogonal matrix, i.e., $\mathbf{C}\mathbf{C}' = \mathbf{I}$. By using matrix $\mathbf{C}$, we can rewrite Eq. (3)(4) into one equation :

$$W = (a_0, d_0, a_1, d_1) = (x_0, x_1, x_2, x_3)\mathbf{C}$$

$$\|AX_1\|^2 + \|DX_1\|^2 = a_0^2 + a_1^2 + d_0^1 + d_1^2 = \|W\|^2$$

$$\|W\|^2 = WW' = X\mathbf{C}\mathbf{C}'X' = XX' = \|X\|^2$$

∎

Lemma 3.10: $\|X\|^2 = \|AX_k\|^2 + \sum_{i=1}^{k} \|DX_i\|^2$.

Proof: By lemma 3.9, $\|X\|^2 = \|AX_1\|^2 + \|DX_1\|^2 = \|AX_2\|^2 + \|DX_2\|^2 + \|DX_1\|^2 = \cdots$. ∎

3.3 Algorithms for trends and surprises mining

Given a time-series graph, at the first glance, one can observe its trend, which is the general direction in which the time-series tends to move. Trends mining strives to obtain the tendency of movement from the time-series data. The multi-level trends are trends in different levels of details. For example, the monthly trends have more details than annual trends. The algorithm for trends mining is shown in Figure 3(a), and a result is shown in Figure 6. It is important to note that when the trend queries have ranges that do not match with node boundaries, we need to retrieve the closest parent node whose boundaries contain the query boundaries, then, apply the algorithm shown in Figure 3(a). For example, if X is daily temperature recording, AX_2 and AX_3 will contain trends for 4-day and 8-day temperature, respectively. If a query is submitted for weekly average temperature, we retrieve AX_2 and compute TX_2 by applying the algorithm of Figure 3(a). From TX_2 we can compute weekly average, which according to our experiments (see Figure 11) is almost as accurate as the result obtaind by using original time series. The same argument holds for surprise queries.

Time-series observations are sometimes influenced by interruptive events, such as strikes, outbreaks of war, sudden political or economic crisis, unexpected heat or cold waves. The consequences of these interruptive events create unusual observations that are inconsistent with the rest of the data. As with many important concepts, what defines a surprise is difficult to specify. This is because an unusual event requires a complete specification of a statistical model. Similar to multi-level trends, multi-level surprises are surprises in different levels of abstraction. For example, an unusual change of temperature in the previous week is a "short term" surprise, while an unusual change of temperature in the previous year is a "long term" surprise. The algorithm for surprises mining is shown in Figure 3(b). Our definition of surprises can be formalized as sudden changes in the original time series data, which are captured by local maximums of absolute value of SX_is (see Figure 7).

4. Reducing the size of TSA-tree

Trivially, the most efficient way for mining trend and surprise sequences is to store the entire TSA-tree on disks. However, the size of the entire TSA-tree is approximately three times the size of the original time-series (assume we split all the way down until the size of nodes becomes 1). When the original size is 2^n, the following equation computes the size of TSA-tree:

$$2^n + 2 \times (1 + 2 + 2^2 + \cdots + 2^{n-1}) = 3 \times 2^n - 2 \quad (5)$$

Hence, due to disk space limitation, we might not be able to maintain the entire TSA-tree disk-resident. That is, part

```
TX_i = Trends - Mining(TSA - Tree, i)
  begin
    fetch AX_i from TSA - Tree;
    TX_i = AX_i;
    for level = i down to 0
        Temp = Up - Sample(TX_i);
        TX_i = Conv(Temp, H_s);
    end
  end
```

(a) Algorithm for trends mining

```
SX_i = Surprises - Mining(TSA - Tree, i)
  begin
    fetch DX_i from TSA - Tree;
    SX_i = DX_i;
    for level = i down to 0
        Temp = Up - Sample(SX_i);
        SX_i = Conv(Temp, G_s);
    end
  end
```

(b) Algorithm for surprises mining

Figure 3. Algorithm for trends and surprises mining

of TSA-tree may need to be stored on tertiary devices (e.g., tapes). However, we can use a nice property of TSA-tree where all the internal nodes can be reconstructed from leaf nodes (see Section 3.2). To identify the optimal subtrees given the available disk space SL, we assume that the costs of merge and split operations are negligible as compared to the cost of disk I/O operation, and the cost of accessing/retrieving a TSA-node is directly proportional to the size of the node [5]. In this section, we prove that when the available space SL is greater than the original size of time series, we can always find a subtree to store on disk without introducing any error. Furthermore, when the available storage space is equal to the original size, the optimal subtree is exactly the leaf nodes of TSA-tree, which is denoted as OTSA-tree as in Def. 4.5. We discuss these two cases in Section 4.1. Finally, we consider the case that all the leaf nodes can not be kept on disk. In this case, we propose two methods to reduce the size of OTSA-tree further: OTSA-tree with node dropping and OTSA-tree with selective coefficient dropping, which are discussed in Section 4.2 and Section 4.3, respectively.

4.1 Optimal TSA-tree

In this section, we formally prove that the storage space required to store the optimal subset of TSA-tree (OTSA-tree) is identical to the size of the original time series. To show this, we assume $Acost(S)$ is the cost to access the subtree S, which is directly proportional to $|S|$.

Definition 4.1: *Optimal TSA subtree:* A subtree ST of TSA which fits on the disk (i.e., satisfies $|ST| \leq SL$) and has the minimum $Acost(ST)$. ∎

[5] There is of course, an overhead of disk seek operation to retrieve each node of the tree. Since this overhead is considered as a constant cost added to all the retrievals, it has been ignored in our model.

Lemma 4.2: *The optimal TSA subtree S will store all the leaf nodes before storing any internal node.*

Proof: We prove this lemma by contradiction. Suppose S is an optimal TSA subtree that contains at least one internal node, say AX_i, without including at least one leaf node, say DX_j. Let us construct another TSA subtree called S', by replacing AX_i with its children, AX_{i+1} and DX_{i+1}, in S. Ignoring the negligible cost of merge operation, we can always obtain AX_i from S' with no extra cost as compared to that of S, i.e., for now at least $Acost(S') = Acost(S)$. Furthermore, $|S| = |S'|$. Now we actually show that $Acost(S') < Acost(S)$ by considering the following two cases:

(1) If DX_{i+1} is already in S, then S' can use the extra space to store DX_j (or part of DX_j). Hence, for queries accessing DX_j, the cost will decrease and hence $Acost(S') < Acost(S)$.

(2) If DX_{i+1} is not in S, and since we know that it is now in S' then for queries accessing DX_{i+1}, the cost will decrease and hence $Acost(S') < Acost(S)$.

Therefore, S is not the optimal subtree of TSA. Contradiction. ∎

Utilizing Lemma 4.2, we now cover three scenarios where the available disk space is less than, equal, or greater than the total size of the TSA leaf nodes $|L|$.

Theorem 4.3: *If $SL <= |L|$, then all the nodes in the optimal TSA subtree are leaf nodes.*

Proof: Suppose S is an optimal TSA subtree. If there is an internal node in S, since $SL <= |L|$, so there must exist a leaf node that is not in S. Hence, from Lemma 4.2, S is not optimal, which is a contradiction. ∎

Theorem 4.4: *If $SL = |L|$, then the optimal TSA subtree is exactly identical to the set of all the leaf nodes.*

Proof: Suppose S is an optimal TSA subtree. Since $SL <= |L|$, from Theorem 4.3, we know all the nodes in S will be leaf nodes, so $S \subseteq L$. From $|S| = SL = |L|$, we know that $S = L$, so the optimal TSA subtree S stores all the leaf nodes. ∎

Definition 4.5: If the available storage space is equal to the size of the original time series X (i.e., $|X| = SL$), then the optimal TSA-tree will be exactly identical to the set of all the leaf nodes (see Theorem 4.4). This optimal subtree is defined as OTSA-tree. ∎

Lemma 4.6: *The cost to obtain AX_i by merging any of its disk resident descendants is $|AX_i|$ (independent of the fact that the descendant is immediate or not). However, the cost to obtain AX_i by splitting its disk resident ancestor at k level above is $2^k \times |AX_i|$.*

Proof: The proof is trivial by using deduction. ∎

Theorem 4.7: *If $SL > |L|$, the optimal TSA subtree contains all the leaf nodes, where storing more internal nodes only improves the performance marginally.*

Proof: The first part of the theorem can be derived from Lemma 4.2 and the second part can be derived from Lemma 4.6. ∎

4.2 OTSA-tree with node dropping

Now, for the case that the available space is less than the original time series, we need to reduce the size of OTSA-tree further. By looking at the distribution of wavelet coefficient values of OTSA as shown in Figure 4, where the X-axis is partitioned by DX_i's, most of the data items in leaf nodes are almost zeros, thus, we can reduce the space required for OTSA-tree by dropping those leaf nodes with less energy.

The problem of OTSA-tree with node dropping can be presented formally as:
$Given$: the available disk space SL,
$Find$: a subset S of OTSA **nodes** which can fit in the limited disk space,
$Such\ that$: we minimize the distance from the original series X to the series $\hat{X}$, where $\hat{X}$ is the series reconstructed from S.
A naive approach is to first construct the space of all possible subsets of OTSA-tree that can fit on the disk and then reconstruct a sequence $\hat{X}$ for each subset. Subsequently, we can compare the distances of the sequences between X and all $\hat{X}$'s to find the subset with the minimum distance, which determines the optimal set of leaf nodes we should keep on the disk. Suppose the length of X is n, the number of leaf nodes is $log_2 n$, then, the number of all possible subsets of leaf nodes is $2^{log_2 n} = n$, considering the time for reconstruction, the total complexity is $O(n^2)$. However, our approach has a complexity of $O(n)$ by using the property of wavelet transform, as shown by the following discussion:

Lemma 4.8: *Suppose we reconstruct a new sequence $\hat{X}$ from a subset of $OTSA$ by dropping some nodes. Let us denote the set of dropped nodes as S. The distance from X to $\hat{X}$ is the sum of the $norm^2$ of nodes in set S, i.e., $\|X - \hat{X}\|^2 = \sum_{node \in S} \|node\|^2$*

Proof: Suppose we drop one node DX_i, and reconstruct $\hat{X}$ from the remaining nodes in $LeafNodes$. By lemma 3.8, $split$ is inverse of $merge$. Hence, we know that OTSA-tree of $\hat{X}$ is same as that of X except for DX_i, which is 0 for $\hat{X}$. By lemma 3.10,$\|X - \hat{X}\|^2 = \|DX_i\|^2$. Since the node we chose was arbitrary, and we can drop another node after DX_i, by induction, the lemma is true. ∎

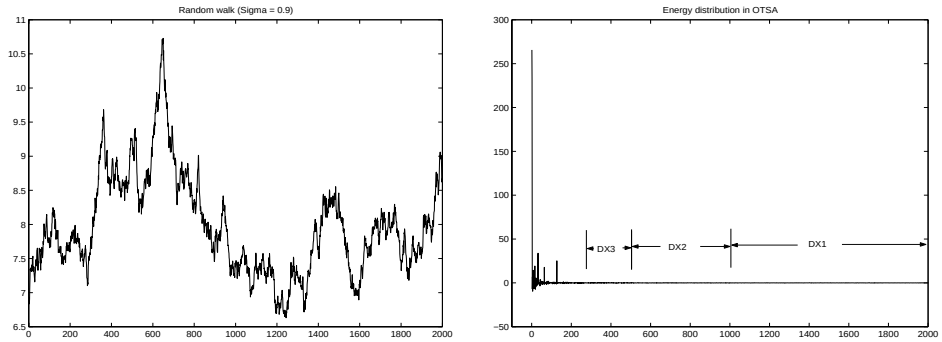
The lemma 4.8 provides us a very desirable property that the error is summable with respect to each node. By this property, the optimal problem can be reduced to the *Fractional Knapsack* problem [11] where a greedy algorithm with $O(n)$ complexity results in the optimal solution [6]. However, we need to define the $value$ and $cost$ for each node of OTSA-tree. The $value$ is directly proportional to the $norm^2$ of a node, and the $cost$ is the size of a node. Hence, the greedy algorithm suggests to start constructing the optimal subtree by choosing the nodes with higher values of $\frac{norm^2(node)}{Size(node)}$. We denote this algorithm as OTSA-w/nd.

4.3 OTSA-tree with coefficient dropping

A drawback with the dropping method proposed in Section 4.2 is that we may lose surprises if the dropped node(s) contains some outliers. In this section, we propose an alternative method to save those outliers separately. We store an outlier as a triplet $(level, i, value)$, which represents the ith data item of $level$ with $value$. To condense further, we consider OTSA-tree as a sequence, with tuples $(position, value)$ instead, where $position$ is the position of a coefficient in the sequence .

Now, the optimality problem is changed to :
$Given$: the available disk space SL,
$Find$: a sub set S of OTSA **coefficients** which can fit in the limited disk space,
$Such\ that$: we minimize the distance from the original series X to the series $\hat{X}$, where $\hat{X}$ is the series reconstructed from S and outliers selected to be stored on disk.
Similar to Lemma 4.8, the error from dropping one data item

[6] To be accurate, for the last node that should be stored partially on the disk, we should store those coefficients with maximum energy values rather than the first few coefficients (more on this in Section 4.3).



(a) Random walk ($\sigma = 0.9$) (b) Coefficients values of leaf nodes

Figure 4. Distribution of wavelet coefficient values in leaf nodes

```
Optimalplacement(OTSA, Space − Limit)
  begin
    Find m, which is the number of coefficients
        afford to store within Space − Limit;
    One pass of OTSA
        find 2m maximum coefficientsinOTSA;
    for i = 1 : Number of OTSA − nodes
      find k_i, the number of coefficients
            in i − th node;
    do
      if(2k_i > |i − th node|)
      store the i − th leaf node;
      (instead of coefficients)
      2k_i = 0
      store the remaining maximum
            coefficients in extra space;
    until any i (2k_i < |i − th node|)
  end
```

Figure 5. OTSA with hybrid coefficient dropping

is equal to the square of the coefficient itself. Hence, the total error is the sum of energy corresponding to each individual coefficient. We again use $FractionalKnapsack$ reduction. The $value$ now is the $norm^2$ of the coefficient, and the $cost$ is the size of the coefficient, which is four bytes per coefficient (float). Therefore, the optimal result should maintain the k coefficients with maximum values on disk. The algorithm for finding the optimal OTSA with coefficient dropping is shown in Figure 5.

This algorithm requires one pass over OTSA coefficients to sort them with the complexity of $O(nlogn)$. Subsequently, it needs a second pass on all OTSA nodes to decide either keeping several coefficients with maximal energy values or the entire leaf node. To eliminate the second pass, we propose two near-optimal heuristics: *OTSA with tail coefficient dropping*(OTSA-w/tcd) and *OTSA with selective coefficient dropping*(OTSA-w/scd). The former keeps the first few coefficients of the entire OTSA and drops the rest. The latter keeps the coefficients with maximum energy in OTSA. A drawback of the latter is that it needs extra storage space to keep the index of the kept coefficients, but it observes a better performance when the time series include many outliers. The optimal algorithm shown in Figure 5 can be considered as a hybrid version of OTSA-w/scd and OTSA-w/tcd. This is because, it switches from one approach to the other depending on the space requirement of each approach.

5. Performance Analysis

In this section, we conducted four sets of experiments to evaluate TSA-tree for various scenarios and applications. First, we graphically verify that our method can be used to visualize trends and surprises at multiple levels of abstractions. For the second set of experiments, we compared our OTSA-tree with DFT [2, 3] and SVD [20] on the accuracy of reconstructing the original time series from the condensed estimations when the storage space is scarce. The experiments showed that even though the complexity of our reconstruction algorithm is less than those of DFT and SVD, the accuracy of our results are comparable. The third set of experiments were performed to demonstrate the superiority of the multi-level structure of OTSA-tree in supporting aggregate queries at multiple levels (e.g., weekly, monthly). Here, we show a significant improvement in I/O operations over SVD due to immediate access to higher level and

hence smaller nodes of OTSA-tree as opposed to the original time series. Finally, we study the impact of increasing the database size and time series length on the performance and query precision of OTSA-tree. The results demonstrate the scalability of our approach.

For all the experiments we either used real time series data from SP500 stock prices of 1998 [22], or synthetic data generated by the random walk algorithm, which is reported as a good model for mimicking stock prices [8]. The reason we used synthetic data is that we did not have access to real time series with length higher than 365 (i.e., one year of stock quotes). Therefore, we fixed the σ parameter of random-walk at 0.9, which we demonstrate it can generate time series consistent in form with our real data (see Section 5.2 for details). In addition, throughout this section, we used standard distortion measure between two time series X and $\hat{X}$, which is computed as $\frac{\|X-\hat{X}\|}{\|X\|}$, as our measure to compute the percentage of error for both query and reconstruction precision.

5.1 Visual verification of trends and surprises mining

Figure 6 shows the multi-level trends on annual copper prices [16] (specifically, levels AX_1 to AX_5). As it is shown in the figure, more details are captured by the lower level TX_1 (see Figure 3(a) for computing TX_i from AX_i), while less details are shown at the higher level TX_5. However, for all the TX_i's, the general trends are the same and they reflect that of the original time series. Figure 7 depicts the effectiveness of our method in capturing surprises at different levels. Similar to Figure 6, lower level SX_1 (see Figure 3(b) for computing SX_i from DX_i) has more details, while higher level SX_5 is more abstract. Again the general surprises for all levels are the same. In Figure 8, we show the trends at multiple levels using two other wavelet filters: Haar in Figure 8(a) and DB2 in Figure 8(b), while in Figure 6 and Figure 7, we had chosen DB4 as the wavelet filter (see [12] for references of DB2 and DB4). Different wavelet filters have different extent of smoothness and complexity. Users can choose different wavelet filters depending on the required/tolerable smoothness and complexity for trends and surprises.

5.2 Space vs. accuracy

In this section, we compare the percentage of error in reconstructing M original time series with length N from their corresponding condensed versions. The condensed versions were generated by four different techniques: DFT, SVD, OTSA-w/tcd and OTSA-w/scd. We varied the length of time series, N, from 252 up to 2048.

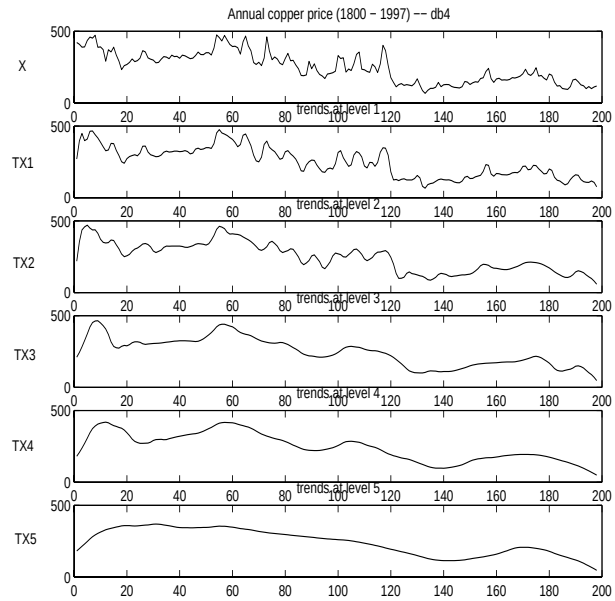


Figure 6. Visualization of multi-level trends (using DB4 wavelet)

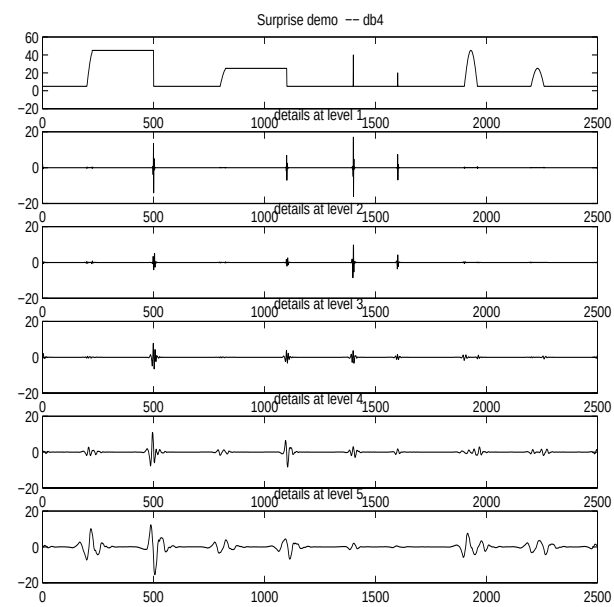
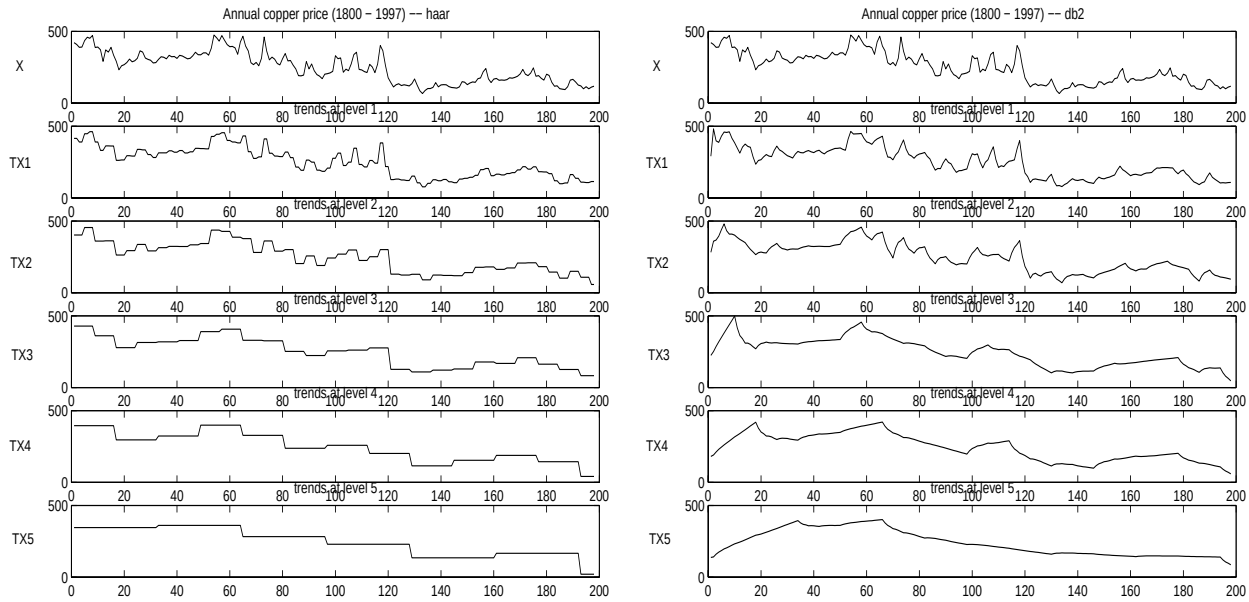


Figure 7. Visualization of multi-level surprises



(a) With Haar wavelet

(b) With DB2 wavelet

Figure 8. Multi-level trends with alternative wavelet filters

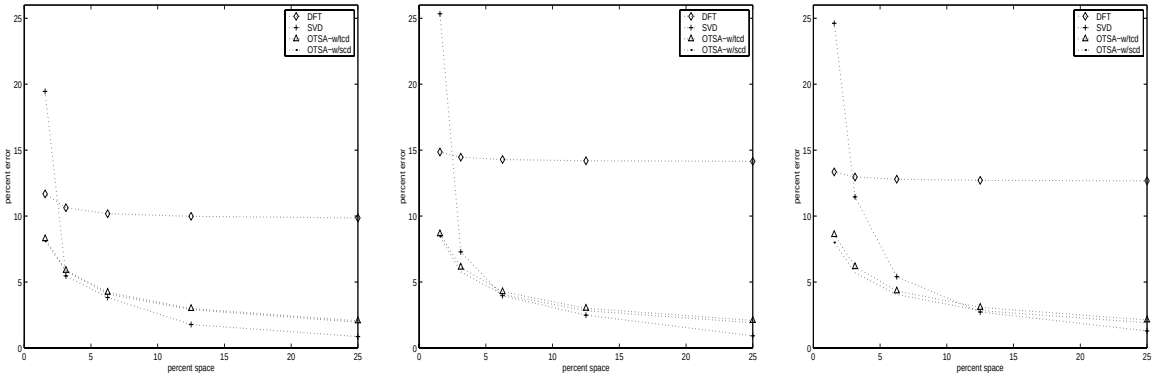
The results are shown in Figure 9 and 10. The X-axis represents the available space which is a percentage of the size of the database (i.e., $M \times N$), and the Y-axis is the percentage of the observed reconstruction error. In Figure 9(a), we used $M = 20$ real time series data from our SP500 source. The length of these time series were $N = 252$. The results show the superiority of OTSA-w/scd over the others. SVD performs poorly at the beginning when the available space is very limited due to the fact that it keeps a few coefficients diagonally in the $M \times N$ matrix of time series data. To perform the same experiment on larger values of M and N, we failed to find real time series data. Therefore, we used random walk to generate synthetic data. In order to be fair, as shown in Figure 9(b), we strive to mimic the real data using random walk. We determined $\sigma = 0.9$ can generate the most consistent results to those of real data.

Thus, for the remainder of our experiments, we fixed σ at 0.9, increased M to 100, and varied N from 512 to 2048. The results are summarized in Figs. 10(a)-(c). In sum, the same trend has been observed for DFT and SVD. An interesting observation is that while OTSA-w/scd continued to be sufficiently and consistently accurate, OTSA-w/tcd also performed well. This is because for large time series, the choice of which coefficients to keep becomes less critical. Therefore, OTSA-w/tcd benefits from the fact that it saves space by not keeping the index of the kept coefficients. It is important to note that the complexity of reconstruction for SVD and DFT are $O(N \times M \times k)$ and $O(M \times N \times log N)$, respectively, where k is the number of kept coefficients. This is while with OTSA-w/tcd, the complexity is only $O(M \times N)$.

OTSA-w/scd started to consistently outperform SVD at $N = 2048$, because when $M << N$, most of the redundancy of data is within the time series, and not across the time series. To illustrate, consider the extreme case, when $M = 1$, $N = 100$, SVD can not reduce the data volume because the reduced diagonally matrix must have at least one entry. However, OTSA can find the correlation across the time dimension.

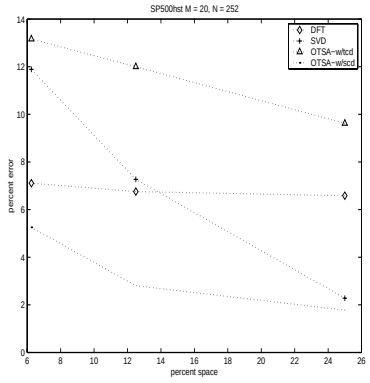
5.3 The advantages of multi-level support

Through the previous set of experiments we demonstrated that under identical storage limitations OTSA if not better, is at least as accurate as any other technique in reconstructing the original time series from its reduced version. In addition, it can perform the reconstruction faster. However, we originally proposed OTSA not for space reduction but for its superiority in supporting multi-level queries. For example, consider we need to compute the monthly average temperature from a data set. With SVD, DFT and other single-level techniques, the only option is to reconstruct the entire time series to support these aggregate queries. Therefore, we not only observe the reconstruction overhead, but also the entire time series need to be retrieved. With OTSA, however, we only need to retrieve a smaller set of OTSA-tree nodes to reconstruct (say) level i trend (AX_i), which reduces both the reconstruction overhead and the retrieval

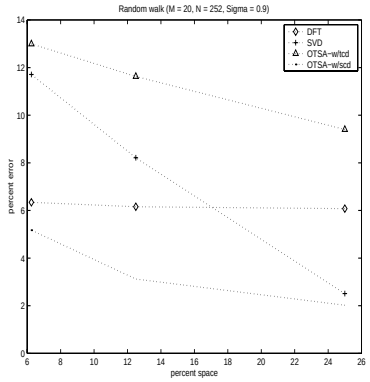


(a) $M = 100, N = 512$ (b) $M = 100, N = 1024$ (c) $M = 100, N = 2048$

Figure 10. Space vs. accuracy



(a) SP500



(b) Random walk ($\sigma = 0.9$)

Figure 9. Real data vs. synthetic data (M = 20, N = 252)

overhead by a factor of $1/2^i$. We of course sacrifice accuracy, but we argue the error is tolerable if the level of the retrieved node is close enough to the query aggregation level[7].

Figure 11 supports our argument. The X-axis is the level of the retrieved node, while the Y-axis is the percentage of query error as compared to the case of retrieving the original time series to answer the query. We assume that OTSA-w/nd is used where only level 3 and above leaf nodes have been stored (i.e., a compression ratio of 12.5%). We performed the experiment for 100 synthetic time series data ($\sigma = 0.9$) with length of 2048 and averaged out the results to reduce the dependability of the experiment from the form of the data set. From Figure 11, we can see that the higher the aggregate level, the lower the percentage of errors. This supports the fact that for queries with high aggregation level (e.g., annual), we do not need to retrieve the entire data, and only some of the TSA nodes at higher levels can answer such queries with high precision.

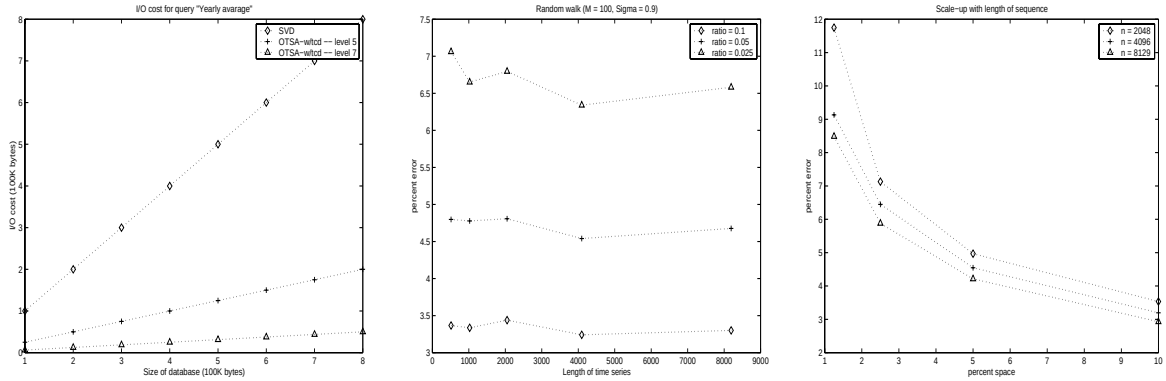
Figure 12 demonstrates the I/O gain of OTSA as compared to SVD to answer an annual average query. The significant gain comes from the fact that OTSA-w/tcd retrieves only AX_5 or AX_7.

5.4 Scalability and analysis

Finally, for the sake of completeness, Figure 12(a,b) demonstrates that our techniques scale up as we increase the database size (i.e., number of time series), and the length of the time series. In these experiments we used synthetic data ($\sigma = 0.9$) and OTSA-w/tcd.

Figure 12(b,c) show the reconstruction error as a function of compression ratio and length of time series. The results demonstrates that as we increase the length of time series, OTSA-w/tcd becomes more accurate.

[7] The same argument applies to surprise queries.



(a) With number of time series ($N = 2048$) (b) With length of time series ($M = 100$) (c) With compression ratio ($M = 100$)

Figure 12. Scale up experiments

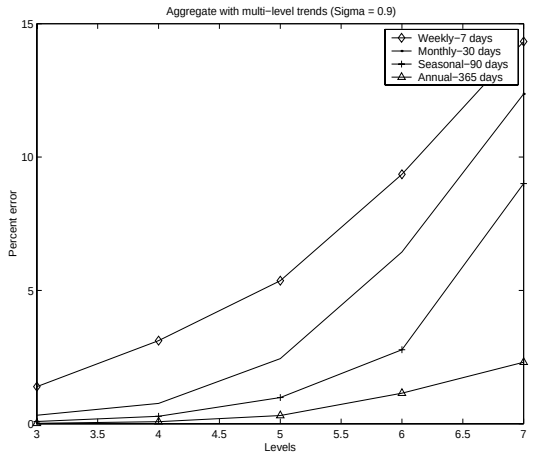


Figure 11. Aggregate queries with multi-level trends

Finally, we compare OTSA with DFT and SVD on the initial time required to condense a time series and the time required to update the condensed time series. The complexity of DFT and SVD for the initial condensing are $O(MN \times lgN)$ and $O(MN^2)$, respectively. The computation complexity for OTSA-w/tcd and OTSA-w/scd are $O(MN)$ and $O(MN \times lgN)$, respectively. For update and append operations, since DFT and SVD require the processing of the entire data set as well as the new data, their complexity is higher than that of OTSA, which can take advantage of the inherent locality of wavelet transform (e.g. Haar filter) for updates and appends.

6. Conclusions and future directions

We proposed a novel data structure for multi-level trends and surprises mining on time-series data, termed TSA-tree. TSA-tree is constructed by recursively applying wavelet transform on the original time series. We proved that a subtree of TSA-tree, termed OTSA-tree, is equal in size to the original time series and can be used to reconstruct the original time series with no error. In addition, we proposed various techniques to reduce the size of OTSA-tree further while keeping the query and reconstruction precisions at reasonable levels.

We conducted many experiments and compared our approach with SVD and DFT, under $N >> M$ assumption. The results demonstrated that under identical compression ratios (when the storage space is scarce), we can reconstruct M original time series from OTSA-tree both faster and more accurately as compared to SVD and DFT. In addition, the complexity of initial construction and update of OTSA-tree are less than those of SVD and DFT. Finally, the inherent multi-level structure of OTSA-tree results in significant improvement in the performance of multi-level trends and surprises queries.

We intend to extend this work in three directions. First, we are in the process of implementing TSA-tree within our JPL GENESIS project[8]. We are using Informix Universal Server v9.2 as our main database server. The split and merge operations are being implemented within our time-series datablade. Consequently, TSA-trees can be constructed (and updated) automatically, given time-series sequences as inputs. Next, we will use the second generation wavelet transform [29, 30], to better adapt to day, week and month scheme instead of using down/up-sampling by 2. Finally, so far we only studied trends and surprises on temporal data (i.e., time-series). However, with our application, we also need to consider spatial trends and surprises. Therefore, we intend to extend this work to focus on spatial pattern mining and eventually introduce a framework for spatiotemporal trend and surprise mining.

References

[1] P. R. A. Arning, R. Agrawal. A linear method for deviation detection in large databases. *KDD*, pages 164–169, 1996.

[2] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence database. *Fourth International Conference on Foundations of Data Organization and Algorithm*, 1993.

[3] R. Agrawal, K.I.Lin, H. S. Sawhney, and K. Sim. Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-series Database. *VLDB*, 1995.

[4] V. Barnett and T. Lewis. *Outliers in Statistical Data*. Wiley & Sons, Chichester, New York, 3rd edition, 1994.

[5] C. S. Burrus, R. A. Gopinath, and H. Guo. *Introduction to wavelets and wavelet transforms : a primer*. Prentice Hall, Upper Saddle River, N.J., 1998.

[6] C.Faloutsos, M.Ranganthan, and Y.Manolopoulos. Fast Subsequence Matching in Time-series Datebase. *SIGMOD, Proc. of Annual Conference, Minneapolis*, 1994.

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

[8] C. Chatfield. *The Analysis of Time Series: an Introduction*. Chapman and Hall, London, 5th edition, 1996.

[9] C. K. Chui. *Wavelets : a tutorial in theory and applications*. Academic Press, Boston, 1992.

[10] C. K. Chui. An overview of wavelets. In *Approximation Theory and Functional Analysis*. Academic Press, Boston, 1993.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. The MIT Press, Cambridge, Mass, 1989.

[12] I. Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 41(7):909–996, October 1988.

[13] D.Rafiei and A. Mendelzon. Similarity-Based Query for Time Series Data. *SIGMOD*, July 1996.

[14] W. A. Fuller. *Introduction to statistical time series*. J. Wiley series in probability and statistics, New York, 2nd edition, 1996.

[15] A. Haar. Zur theorie der orthogonalen funktionensysteme. *Mathematics Annal.*, 69:331–371, 1910.

[16] R. Hyndman. http://www-personal.buseco.monash.edu.au/ hyndman/tsdl/.

[17] M. Kendall. *Time Series*. Griffin, London, 2nd edition, 1976.

[18] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. *Proceedings of the 24th VLDB Conference*, pages 392–403, 1998.

[19] E. M. Knorr and R. T. Ng. Finding intensional knowledge of distance-based outliers. *Proceedings of the 25th VLDB Conference*, pages 211–222, 1999.

[20] F. Korn, H. V. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. *Proceedings of the ACM SIGMOD international conference on Management of data*, 26(2):289–300, 1997.

[21] C. Li, P. S. Yu, and V. Castelli. Malm: A framework for mining sequence database at multiple abstraction levels. In *Proceedings of the 1998 ACM 7th international conference on Information and knowledge management*, pages 267–272, 1998.

[22] B. A. Logue. http://www.mdshistoricalquotes.com/.

[23] S. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE transactions on pattern Analysis and Machine Intelligent*, 11(7):674–693, July 1989.

[24] Y. Qu, C. Wang, and X. S. Wang. Supporting fast search in time series for movement patterns in multiple scales. In *Proceedings of the 1998 ACM 7th international conference on Information and knowledge management*, pages 251–258, 1998.

[25] D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. *SIGMOD*, pages 13–24, 1997.

[26] R.Agrawal, G. Psaila, E. L. Wimmers, and M. Zait. Querying Shapes of Histories. *VLDB*, September 1995.

[27] R.D.Edwards and J.Magee. *Technical analysis of stock trends*. John Magee, Springfield, Massachsetts, 1969.

[28] H. Shatkay and S. Zdonik. Approximate queries and representations for large data sequence. *ICDE*, 1996.

[29] W. Sweldens. The lifting scheme: A new philosophy in biorthogonal wavelet construction. In A. F. Laine and M. Unser, editors, *Wavelet Applications in Signal and Image Processing III*, pages 68–79, 1995.

[30] W. Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal.*, 29(2):511–546, 1997.

[8] see http://dimlab.usc.edu/ for a demonstration.