Finding Surprising Patterns in a Time Series Database in Linear Time and Space

Eamonn Keogh
eamonn@cs.ucr.edu

Stefano Lonardi
stelo@cs.ucr.edu

Bill 'Yuan-chi' Chiu
ychiu@cs.ucr.edu

Department of Computer Science and Engineering
University of California
Riverside, CA 92521

ABSTRACT

The problem of finding a specified pattern in a time series database (i.e. query by content) has received much attention and is now a relatively mature field. In contrast, the important problem of enumerating all surprising or interesting patterns has received far less attention. This problem requires a meaningful definition of "surprise", and an efficient search technique. All previous attempts at finding surprising patterns in time series use a very limited notion of surprise, and/or do not scale to massive datasets. To overcome these limitations we introduce a novel technique that defines a pattern surprising if the frequency of its occurrence differs substantially from that expected by chance, given some previously seen data. This notion has the advantage of not requiring an explicit definition of surprise, which may be impossible to elicit from a domain expert. Instead the user simply gives the algorithm a collection of previously observed normal data. Our algorithm uses a suffix tree to efficiently encode the frequency of all observed patterns and allows a Markov model to predict the expected frequency of previously unobserved patterns. Once the suffix tree has been constructed, a measure of surprise for all the patterns in a new database can be determined in time and space linear in the size of the database. We demonstrate the utility of our approach with an extensive experimental evaluation.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

Keywords

Time series, Suffix Tree, Novelty Detection, Anomaly Detection, Markov Model, Feature Extraction.

SIGKDD '02, July, 23-26 2002, Edmonton, Alberta, Canada.

1. INTRODUCTION

The problem of finding a specified pattern in a time series database (i.e. query by content) has received much attention and is now a relatively mature field [12, 22, 19, 21]. In contrast, the problem of enumerating all surprising or interesting patterns has received far less attention. The utility of such an algorithm is quite obvious. It would potentially allow a user to find surprising patterns in a massive database without having to specify in advance what a surprising pattern looks like.

Note that this problem should not be confused with the relatively simple problem of outlier detection. Hawkins' classic definition of an outlier is "*... an observation that deviates so much from other observations as to arouse suspicion that it was generated from a different mechanism*" [18]. However we are not interested in finding individually surprising datapoints, we are interested in finding surprising patterns, i.e., combinations of datapoints whose structure and frequency somehow defies our expectations. The problem is referred to under various names in the literature, including novelty detection [9, 5] and anomaly detection [37].

The problem requires a meaningful definition of "surprise". The literature contains several such definitions for time series; however they are all too limited for a useful data-mining tool. Consider for example the notion introduced by Shahabi *et al.* [32]. They define *surprise* in time series as "*...sudden changes in the original time series data, which are captured by local maximums of the absolute values of (wavelet detail coefficients)*". However it is not difficult to think of very surprising patterns that defy this rule. For example, consider Figure 1. Here the beginning of each normal heartbeat is considered very surprising, but the temporary absence of a heartbeat is considered to be the least surprising subsection of the time series!

Several other definitions of surprise for time series exist, but all suffer from similar weaknesses [7, 37, 38, 9]. To overcome these limitations we introduce a novel definition that defines a pattern surprising if the frequency of its occurrence differs substantially from that expected by chance, given some previously seen data. This no-

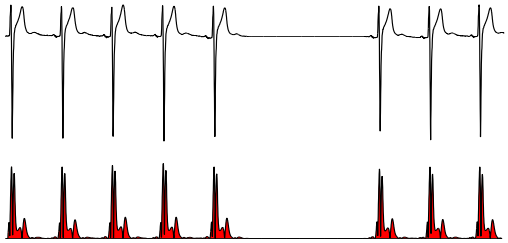

Figure 1: The time series at the top is a normal healthy human electrocardiogram with an artificial "flatline" added. The sequence at the bottom indicates how surprising local subsections of the time series are under the measure introduced in Shahabi *et al.*. In this case the beginning of each normal heartbeat is very surprising, but the "flatline" is the least surprising part of the time series, a very unintuitive result

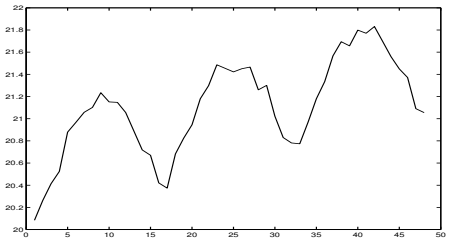

Figure 2: An example of the classic "head and shoulders" pattern

tion has the advantage of not requiring an explicit definition of surprise, which may in any case be impossible to elicit from a domain expert. Instead the user simply gives the algorithm a collection of previously observed data, which is considered normal. The measure of surprise of a newly observed pattern is considered relative to this data collection, and thus eliminates the need for a specific model of normal behavior.

Note that unlike all previous attempts to solve this problem, the measure of surprise of a pattern is not tied exclusively to its structure. Instead it depends on the departure of the frequency of the pattern from its expected frequency. This is the crucial distinction of our approach from all the others.

For example consider the "head and shoulders" pattern shown in Figure 2. The existence of this pattern in a stock market time series should not be considered surprising since it is known to occur (even if only by chance). However, if it occurred ten times this year, as opposed to occurring an average of twice a year in previous years, our measure of surprise will flag the pattern as being surprising. Intuitively, the pattern would also be surprising if its frequency of occurrence is less than expected. Once again our definition would flag such patterns.

Our definition of surprise would be of little utility to the data mining community without a technique that allowed efficient determination of the expected frequency of a pattern. We demonstrate how a suffix tree can be used to efficiently encode the frequency of all observed patterns. Since it is possible that a pattern observed in the new data was not observed in the training data, we demonstrate a technique based Markov models to calculate the expected frequency of previously unobserved patterns. Once the suffix tree has been constructed, the measure of surprise for all the patterns in a new database can be determined in time linear in the size of the database.

The rest of the paper is organized as follows. In Section 2 we discuss feature extraction techniques to discretize time series, a necessary preprocessing step for our approach. In Section 3 we introduce some background material on string processing, including suffix trees and Markov models. In Section 4 we show how to efficiently compute any pattern's expected frequency by using suffix trees. Having reviewed the materials discussed in previous sections, we finally introduce our algorithm in Section 5. In Section 6 we demonstrate the utility of our approach with a detailed empirical evaluation. We wait until Section 7, when the reader's intuitions about the problem are more fully developed to discuss related work. Finally in Section 8 we offer conclusions and directions for future research.

2. DISCRETIZING TIME SERIES

For concreteness we more formally define our intuition of surprise as follows.

DEFINITION 2.1. *A time series pattern P, extracted from database X is surprising relative to a database R, if the frequency of its occurrence is greatly different to that expected by chance, assuming that R and X are created by the same underlying process.*

In order to compute this measure, we must calculate the probability of occurrence for the pattern of interest. Here we encounter the familiar paradox that the probability of a particular real number being chosen from any distribution is zero [15]. Since a time series is an ordered list of real numbers the paradox clearly applies. The obvious solution to this problem is to discretize the time series into some finite alphabet Σ. Using a finite alphabet allows us to avail of Markov models to estimate the expected probability of occurrence of a previously unseen pattern.

The problem of discretizing time series into a finite alphabet has received much attention in diverse fields, including astronomy, medicine, chemistry, etc. (See [10] for an exhaustive overview). The representation has also captured the attention of the data mining community who use discretized time series to support similarity

Symbol	
R	the reference time series database (consisting of real numbers)
X	the time series database (to be mined for surprising patterns)
r	the discrete version of R
x	the discrete version of X
l_1	the feature window length
l_2	sliding window length
a	the alphabet size

Table 1: A summary of the major notation use in the work. More complete definitions are given in the relevant sections

search [19] and to enable change point detection [16].

Below we give a generic algorithm to discretize a time series dataset such that each symbol is equiprobable. A table of notation used in this, and subsequent algorithms is given in Table 1.

The inputs are a reference time series database R, the feature window length and the size of the desired alphabet. The feature window length is the length of a sliding window that is moved across the time series. At each time step, the portion of data falling within the window is examined, and a single real number, describing some feature of the data is extracted. After the features have been extracted, they are sorted so the boundaries that contain an equal number of extracted features can be determined. At this point the unsorted features are scanned, each feature is tested to see which range it maps to, and matching symbol is assigned. An outline of the algorithm is shown in Table 2.

Note that the one element of the algorithm we did not specify is the EXTRACT_FEATURE subroutine. Here we have been deliberately vague. The best feature extraction technique may be domain dependent. Possible features include the mean of the data [21], the slope of the best-fitting line [16, 22], the second wavelet coefficient[1], the second real Fourier coefficient [12], etc. For simplicity we will consider only the slope of the best-fitting line for the rest of this paper.

We also have not stated how the two parameters, the feature window length and the size of the desired alphabet, are chosen. As emphasized in [8] and elsewhere, data mining is an iterative activity, and "discovery algorithms should be run several times with different parameter settings". Alternatively, techniques that use maximum entropy based methods can be used to decide reasonable parameters to discretize time series [28].

The time complexity for the above algorithm is dominated by the need to sort the features to allow determination of the feature boundaries. However these

[1]The first wavelet coefficient, and the first real Fourier coefficient, simply contain the mean of the signal.

```
string DISCRETIZE_TIME_SERIES (time_series X,
        int l1, int a)
   for i = 1, |X| − l1 + 1
      let features[i] = EXTRACT_FEATURE(X[i,i+l1])
   let sorted_features = SORT(features)
   for j = 1, a
      let pointer = j |features| / a
      let boundaries[j] = sorted_features[pointer]
   for i = 1, |features|
      let x[i] =
         MAP_REAL_TO_INT(boundaries, features[i])
   return x
```

Table 2: Outline of the algorithm for the discretization of the time series: t is the time series data, l_1 is the feature window length, a is the alphabet size

feature boundaries are very stable, and can be reliably estimated from a subsample of the data [8]. For large databases we can determine the feature boundaries from a subsample of size $s = \sqrt{|R|}$ [8]. Since $s \log(s) < |R|$, the feature extraction algorithm is $O(|R|)$.

We have included this generic feature extraction for completeness, however the rest of our work does not require any particular feature extraction approach. A multitude of alternative methods to discretize the time series could be used; including vector quantization [23] and clustering based techniques [8].

3. BACKGROUND ON STRING PROCESSING

We use Σ to denote a nonempty *alphabet* of *symbols*. A *string* over Σ is an ordered sequence of symbols from the alphabet. Given a string x, the number of symbols in x defines the *length* $|x|$ of x. Henceforth, we assume $|x| = n$. The empty string has length zero, and is denoted by ϵ.

Let us decompose a text x in uvw, i.e., $x = uvw$ where u, v and w are strings over Σ. Strings u, v and w are called *substrings*, or *words*, of x. Moreover, u is called a *prefix* of x, and w is called a *suffix* of x.

We write $x_{[i]}$, $1 \leq i \leq |x|$ to indicate the i-th symbol in x. We use $x_{[i,j]}$ as shorthand for the substring $x_{[i]}x_{[i+1]} \ldots x_{[j]}$ where $1 \leq i \leq j \leq n$, with the convention that $x_{[i,i]} = x_{[i]}$. Substrings in the form $x_{[1,j]}$ corresponds to the prefixes of x, and substrings in the form $x_{[i,n]}$ to the suffixes of x.

We say that a string y has an *occurrence* at position i of a text x if $y_{[1]} = x_{[i]}$, $y_{[2]} = x_{[i+1]}$, $\ldots$, $y_{[m]} = x_{[i+m-1]}$, where $m = |y|$. For any substring y of x, we denote by $f_x(y)$ the number of occurrences of y in x.

Throughout this document, variables y and w usually indicate substrings of the text x. Unless otherwise spec-

ified, we assume the generic term m as the length of any of these words.

3.1 Markov models

We consider a string generated by a stationary Markov chain of order $M \geq 1$ on the finite alphabet Σ. Let $x = x_{[1]}x_{[2]}\ldots x_{[n]}$ be an observation of the random process and $y = y_{[1]}y_{[2]}\ldots y_{[m]}$ an arbitrary but fixed pattern over Σ with $m < n$.

The stationary Markov chain is completely determined by its *transition* matrix $\Pi = (\pi(y_{[1,M]}, c))_{y_{[1]},\ldots,y_{[M]},c\in\Sigma}$ where

$$\pi(y_{[1,M]}, c) = \mathbf{P}(X_{i+1} = c \,|X_{[i-M+1,i]} = y_{[1,M]})$$

are called *transition probabilities*, with $y_{[1]}, \ldots, y_{[M]}, c \in \Sigma$ and $M \leq i \leq n-1$. The vector of the *stationary probabilities* μ of a stationary Markov chain with transition matrix Π is defined as the solution of $\mu = \mu\Pi$.

We now introduce the random variable which describes the occurrences of the word y. We define $Z_i, 1 \leq i \leq n-m+1$ to be 1 if y occurs in x starting at position i, 0 otherwise. We set

$$Z_y = \sum_{i=1}^{n-m+1} Z_{i,y}$$

so that Z_y is the random variable for the total number of occurrences $f_x(y)$.

In the stationary M-th order Markovian model the expectation of Z_i, which represents the probability that y occurs at a given position i, is given by

$$\begin{aligned} E(Z_i) &= \mathbf{P}(X_{[i,i+m-1]} = y) \\ &= \mu(y_{[1,M]})\pi(y_{[1,M]}, y_{[M+1]})\pi(y_{[2,M+1]}, y_{[M+2]}) \\ &\quad \cdots \pi(y_{[m-M,m-1]}, y_{[m]}) \\ &= \mu(y_{[1,M]}) \prod_{i=1}^{m-M} \pi(y_{[i,i+M-1]}, y_{[i+M]}). \end{aligned}$$

The expected count of the occurrences y under the Markov model is therefore

$$\begin{aligned} E(Z_y) &= (n-m+1)E(Z_i) \qquad (1) \\ &= (n-m+1)\mu(y_{[1,M]}) \prod_{i=1}^{m-M} \pi(y_{[i,i+M-1]}, y_{[i+M]}) \end{aligned}$$

because the distribution of the Z_i's does not depend on i.

When the true model is *unknown*, the transition and stationary probabilities have to be estimated from the observed sequence x. Let y be a substring of x, where $m = |y| \geq M+2$. The transition probability can be estimated by the *maximum likelihood estimator* [30]

$$\hat{\pi}(y_{[1,M]}, c) = \frac{f_x(y_{[1,M]}c)}{f_x(y_{[1,M]})} \qquad (2)$$

and the *stationary probability* by the maximum likelihood estimator

$$\hat{\mu}(y_{[1,M]}) = \frac{f_x(y_{[1,M]})}{n-M+1}. \qquad (3)$$

Substituting in equation (1) for the estimators (2) and (3) we obtain an estimator of the expected count of y

$$\hat{E}(Z_y) = \frac{\prod_{i=1}^{m-M} f_x(y_{[i,i+M]})}{\prod_{i=2}^{m-M} f_x(y_{[i,i+M-1]})}.$$

A precise relationship between the expectation of y and the expectation of its prefix and suffix is established in the following fact.

LEMMA 3.1. *Let y be a substring of x and $w_1 = y_{[2,m]}$, $w_2 = y_{[1,m-1]}$. Then*

$$\hat{E}(Z_y) = \frac{f(y_{[1,M+1]})}{f(y_{[2,M+1]})}\hat{E}(Z_{w_1}) = \hat{E}(Z_{w_2})\frac{f(y_{[m-M,m]})}{f(y_{[m-M,m-1]})}$$

3.2 Suffix Trees

A naive and simple method to count the number of occurrences of each substring in a sequence is to create a look-up table. The table has an entry for each word. Given a word w, a one-to-one hash function returns the index in the table, as follows

$$h(y) = \hat{h}(w_{[1]}) + \hat{h}(w_{[2]})|\Sigma| + \cdots + \hat{h}(w_{[m]})|\Sigma|^{m-1}$$

where $\hat{h}$ is a one-to-one map from symbols in Σ to the range $[0, \ldots, |\Sigma|-1]$. Filling in the entries of the table takes $O(nm)$, where m is the length of the substring we want to consider. However, the space needed is $O(|\Sigma|^m)$ and it requires at least $O(|\Sigma|^m)$ to be initialized. Note that $|\Sigma|^m$ can be much larger than nm. Indeed, when $m > \log_{|\Sigma|} n$ the large majority of the strings in Σ^m do not appear at all in x. A query in the table takes only constant time.

The hash table is a convenient data structure as long as m is bounded by a relatively small constant. If we allow m to grow as a function of n, for example $m \propto log(n)$, then the time and space required to build the hash table would be *exponential* in the size of the input.

A more space-efficient data structure to organize a dictionary of words is to use a *suffix tree* (see, e.g., [17] and references therein). The suffix tree is a type of digital search tree that represents a set of strings over a finite alphabet Σ. It has n leaves, numbered 1 to n. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty substring of x. No two edges outgoing from a node can have labels beginning with the same character. The tree has the property that for any leaf i, the concatenation of the labels on the path from the root the the leaf i spells out exactly the suffix of x that starts at position

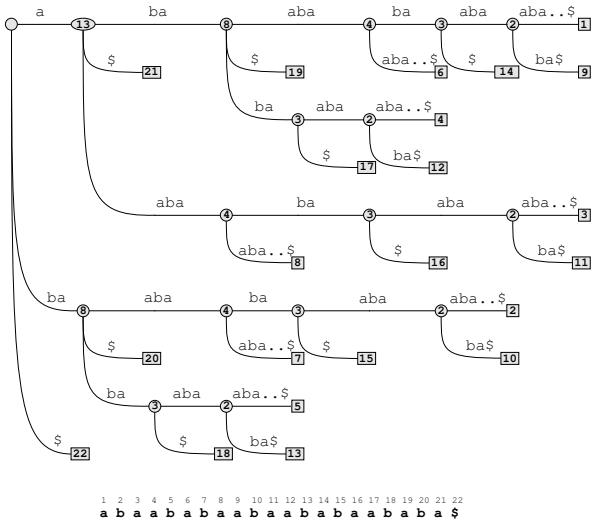

Figure 3: The suffix tree T_x for the string $x =$ abaababaabaabababaaba$, with internal nodes storing the number of occurrences

i, that is $x_{[i,n]}$. The substrings of x can be obtained by spelling out the words from the root to any internal node of the tree or to any position in the middle of an edge.

In order to achieve overall linear-space allocation, the labels on the edges are described implicitly: for each word, it suffices to save an ordered pair of integers indexing one of the occurrences of the label in the text. Each edge label requires thus constant space, which, in conjunction with the fact that total number of nodes and edges is bounded by $O(n)$, results in the overall linear space for the tree.

Several clever $O(n \log |\Sigma|)$ constructions are available (see, e.g., [27, 36]). More recent linear-time algorithms are by Ukkonen [34] which is on-line, and by Farach [13] which is optimal for large alphabets. The large majority of these constructions exploit the presence of *suffix links* in the tree. The existence of suffix links is based on the following fundamental fact.

LEMMA 3.2. *If $w = ay, a \in \Sigma$ has a proper locus in T_x, then so does y.*

Accordingly, suffix links are maintained in the tree from the locus of each string ay to the locus of its suffix y, for all $a \in \Sigma$.

Having built the tree, some additional processing make it possible to count and locate all the distinct instances of any pattern in $O(m)$ time, where m is the length of the pattern. In fact, the computation of the statistics of all substrings of a string is a direct application of suffix trees. We first need some definitions. We define the *leaf-list* $LL(u)$ of a node u as the ordered set of indices stored in the leaves of the subtree rooted at u. We refer to the unique string on the path from the root to a node u of the tree as the *path-label* $L(u)$ of u. Vertex u is also called the *proper locus* of $L(u)$. Some strings do not have a proper locus because their paths end in the middle of an arc.

Given a word w, we denote by $<w>$ its proper locus, if it exists. If instead w ends in the middle of an arc then $<w>$ denotes the node corresponding to the shortest extension of w that has a proper locus. Clearly, $L(<w>) = w$.

By the structure of a suffix tree, the number of occurrences $f_x(w)$ of any string w is given by the number of leaves in the subtree rooted at $<w>$, that is, $f(w) = |LL(<w>)|$. In Figure 3, the number of occurrences is stored in the internal nodes. The algorithm that annotates the tree with the value $f(w)$ takes linear time and space in the size of x.

4. COMPUTING SCORES BY COMPARING TREES

Let r be the reference sequence, and x the sequence under analysis. A preprocessing phase takes care of annotating the suffix tree T_x with the scores of each substring of x. Although the algorithm does not require to bound the size of the substrings, it is reasonable to assume that we will never consider substrings longer that $\log_{|\Sigma|} n$ symbols. It is well known that words longer than $\log_{|\Sigma|} n$ symbols have an expected count which tend to a constant instead of growing to infinity when n goes to infinity. They are therefore of little interest to us.

In the first step of the preprocessing phase we build the trees T_r and T_x and we annotate the internal nodes of both trees with the number of occurrences. This step requires linear time and space.

In the second step of preprocessing, we visit in a breadth-first order each node u of T_x. For each string $w = L(u)$ we search for the node $<w>$ in T_r, if it exists. In the case it exists, we compute directly the score, assuming $\alpha f_r(w)$ to be the expected number of occurrences of w in the reference string, where $\alpha = \frac{|x|-m+1}{|r|-m+1}$. The scale factor α takes care of of adjusting the occurrences based on the length of x and r. For example, if r is two times longer than x we scale the number of occurrence observed in r by roughly $1/2$.

If otherwise the substring w does not occur in T_r then we look for the largest l in the interval $[1, \ldots, |w| - 1]$ such that all the strings $w_{[j,j+l]}$ occur in T_r, for $j = 1, \ldots, |w| - l$. In other words, we look for the longest set of strings from T_r that cover w as it is done for the estimator of the expectation for Markov chains (see Section 3.1). This strategy corresponds to the idea of trying first the higher Markov orders, and falling back to

suffix_tree PREPROCESS (**string** r, **string** x)
  let T_r = SUFFIX_TREE(r)
  let T_x = SUFFIX_TREE(x)
  let $\alpha = \frac{|x|-m+1}{|r|-m+1}$
  ANNOTATE_$f(w)(T_r)$
  ANNOTATE_$f(w)(T_x)$
  visit T_x in breadth-first traversal, for each node u **do**
    let $w = L(u), m = |w|$
    if w occurs in T_r **then**
      let $\hat{E}(w) = \alpha f_r(w)$
    else
      find the largest $1 < l < m-1$ such that $\prod_{j=1}^{m-l} f_r(w_{[j,j+l]}) > 0$
      using the suffix tree T_r
      if such l exists **then**
        let $\hat{E}(w) = \alpha \frac{\prod_{j=1}^{m-l} f_r(w_{[j,j+l]})}{\prod_{j=2}^{m-l} f_r(w_{[j,j+l-1]})}$
      else
        let $\hat{E}(w) = (|x|-m+1)\prod_{i=1}^{m} w_{y_{[i]}}$
    let $z(w) = f_x(w) - \hat{E}(w)$
    store $z(w)$ in the node u
  return T_x

Table 3: Outline of the preprocessing algorithm for the computation of the scores obtained comparing the trees of a reference string r against the string under analysis x

lower orders whenever the information to compute the estimator of the expectation are insufficient. If every possible choice does not meet the requirements, we use the probability of the symbols from T_r to compute the estimate.

Finally, we set the surprise $z(w)$ to be the difference between the observed number of occurrences $f_x(w)$ and $\hat{E}(w)$. The preprocessing algorithm is sketched in Figure 3. The time complexity depends on the time taken to compute $\hat{E}(w)$. If the algorithm would be implemented as in Figure 3, the time complexity would be superlinear. To compute efficiently $\hat{E}(w)$ we use Lemma 3.1 and the suffix links of Lemma 3.2. We defer the algorithmic and combinatorial analysis to the journal version of this paper.

5. "TARZAN" ALGORITHM

Having reviewed extensive material on feature extraction, Markov models and suffix trees, we now give a concise description of the proposed algorithm, which we call TARZAN[2]. The basic algorithm is sketched in Table 4.

[2]TARZAN is not an acronym. It is a pun on the fact that the heart of the algorithm relies on comparing two suffix trees, *"tree to tree"*. Tarzan (R) is a registered tradermark owned by Edgar Rice Burroughs, Inc.

void TARZAN (**time_series** R, **time_series** X,
  int l_1, **int** a, **int** l_2, **real** c)
  let x = DISCRETIZE_TIME_SERIES (X, l_1, a)
  let r = DISCRETIZE_TIME_SERIES (R, l_1, a)
  let T_x = PREPROCESS (r, x)
  for $i = 1, |x| - l_2 + 1$
    let $w = x_{[i,i+l_2-1]}$
    retrieve $z(w)$ from T_x
    if $|z(w)| > c$ **then print** $i, z(w)$

Table 4: Outline of the Tarzan algoritm: l_1 is the feature window length, a is the alphabet size for the discretization, l_2 is the scanning window length and c is the threshold

The inputs are the reference database R, the database to be examined X, and the three parameters which control the feature extraction and representation. The algorithm begins by discretizing the data to the desired granularity. The two resultant strings are passed to the PREPROCESS algorithm which constructs the annotated suffix tree T_x. After this has been accomplished, the surprise of each substring found in x can determined. Those substrings which have surprising ratings exceeding a certain user defined threshold (as defined by the absolute value of $z(w)$) can be returned and examined by the user.

The length l_2 of the sliding window is connected with the feature window length l_1 and the alphabet size a (which have been discussed in Section 2). We suggest choosing $l_2 < \log_{|\Sigma|} |x|$ because words longer than $\log_{|\Sigma|} |x|$ have extremely small expectations and belong to a different probabilistic regime. In fact, scores $z(w)$ are asymptotically Gaussian distributed when $|w| < \log_{|\Sigma|} |x|$ and Poisson distributed for longer words [30]. The threshold c can be identified by gathering statistics about the distribution of the scores and/or assuming the distribution of the scores to be normal.

6. EXPERIMENTAL EVALUATION

Empirically we would like to demonstrate two features of our proposed approach.

- **Sensitivity (High True Positive Rate):** the algorithm can find truly surprising patterns in a time series.
- **Selectivity (Low False Positive Rate):** the algorithm does not find spurious "surprising" patterns in a time series.

We compare our approach with the TSA-tree Wavelet based approach of Shahabi *et al.* [32] and to the Immunology (IMM) inspired work of Dasgupta and Forrest [9], which are the only obvious candidates for comparison. More details about these approaches are contained in Section 7. We do not compare the CPU times for each

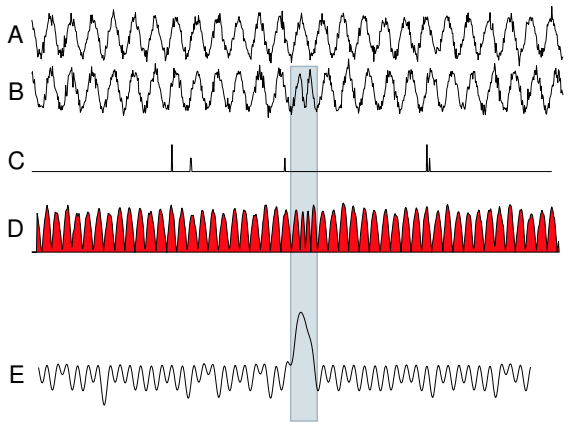

Figure 4: A comparison of three anomaly detection algorithms on the same task. A) The training data, a slightly noisy sine wave. B) A time series containing a synthetic "anomaly", it is a noisy sine wave that was created with the same parameters as the training sequence. Then the period of the sine wave between the 400^{th} and 432^{th} points (denoted by the gray bar) was halved. C) The IMM anomaly detection algorithm failed to find the anomaly, and introduced some false alarms. D) The TSA-Tree approach is also unable to detect the anomaly. E) Tarzan shows a strong peak for the duration of the anomaly

algorithm, since we cannot guarantee that our reimplementation of the rival methods is efficient. However we have already noted that the time complexity of TARZAN is linear in the size of $x + r$.

We begin with a very simple experiment as a reality check. We constructed a reference dataset by creating a sine wave with 800 datapoints and adding some Gaussian noise (each complete sine wave is 32 datapoints long). We then built a test dataset using the same parameters as the reference set, however we also inserted an artificial anomaly by halving the period of the time series in the region between the 400^{th} and 432^{th} datapoints. In other words, that small subsection of the test time series has two short sine waves instead of one. We compared all three approaches under consideration. The results are shown in Figure 4. We used a feature window of length $l_1 = 12$ for TARZAN and IMM, and an alphabet of size $a = 4$ for TARZAN.

The IMM approach was unable to find the anomaly, and it introduced some false alarms. Unlike the two other approaches, this method has a stochastic component. On some runs it did detect the anomaly, but it always produce several false alarms of equal magnitude. The TSA approach also failed to find the anomaly[3]. As

[3]The authors define surprise as an absolute value, but visualize the level of surprise in their paper without taking the absolute value. For clarity and consistency we have graphed absolute values.

Figure 5: The first three weeks of the power demand dataset. Note the repeating pattern of a strong peak for each of the five weekdays, followed by relatively quite weekends

noted by the authors, surprising patterns might exist at different scales, the graphic above shows the results at "level 4" (see original paper for details [32]), however similar results are seen at other levels. In contrast to the other techniques TARZAN shows a strong peak for the duration of the anomaly. Note that for consistency with the other techniques we flipped the results for TARZAN upside down, so the low expectation for the anomaly shows as a peak.

Testing the ability of the algorithms to find surprising patterns on real data is a greater challenge, since the results may be subjective. To address this problem we consider a dataset that contains the power demand for a Dutch research facility for the entire year of 1997 [35]. The data is sampled over 15 minute averages, and thus contains 35,040 points. The nice feature of this dataset is that although it contains great regularity, as shown in Figure 5, it also contains regions that could objectively be said to be surprising or anomalous. In particular, there are several weeks on which one or more days were national holidays, and thus the normal pattern of five weekday peaks, followed by a relatively flat weekend, is disturbed.

We used from Monday January 6^{th} to Sunday March 23^{rd} as reference data. This time period is devoid of national holidays. We processed the remainder of the year with TARZAN, with a window size equivalent to 4 hours ($l_1 = 16$ datapoints), and an alphabet of size $a = 4$. Because of the size of the dataset we will just show the three most surprising sequences found by each algorithm. For each of the three approaches we show the entire week (beginning Monday) in which the three largest values of surprise fell. The results are shown in Figure 6.

Both TSA-tree and IMM returned sequences that appear to be normal workweeks, however TARZAN returned three sequences that correspond to the weeks that contain national holidays in the Netherlands. In particular, from top to bottom, the week spanning both December 25^{th} and 26^{th} and the weeks containing Wednesday April 30^{th} (Koninginnedag, "Queen's Day") and May 19^{th} (Whit Monday) respectively. These results present strong visual evidence that TARZAN is able to find surprising patterns in time series.

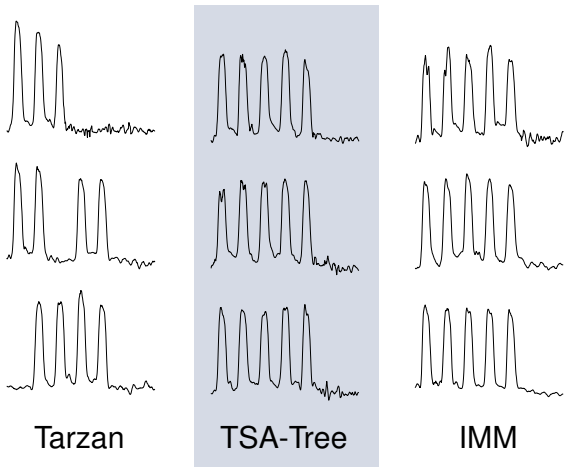

Figure 6: The three most surprising weeks in the power demand dataset, as determined by Tarzan, TSA-Tree and IMM

The previous experiments demonstrate the ability of TARZAN to find surprising patterns, however we also need to consider TARZAN's selectivity. If even a small fraction of patterns flagged by our approach are false alarms, then, as we attempt to scale to massive datasets, we can expect to be overwhelmed by innumerable spurious "surprising" patterns.

In designing an experiment to show selectivity we are faced with the problem of finding a database guaranteed to be free of surprising patterns. Because using a real data set for this task would always be open to subjective post-hoc explanations of results, we will conduct the experiment on random walk data [12, 21].

By definition, random walk data can contain any possible pattern. In fact, as the size of a random walk dataset goes to infinity, we should expect to see every pattern repeated an infinite number of times [15]. We can exploit this property to test our algorithm. Suppose we fix a test dataset X_{RW} to be a random walk dataset of length 128. If we train TARZAN on another short random walk dataset, we should expect that the test data would be found surprising, since the chance that similar patterns exist in the short training database are very small. However as we increase the size of the training data, the overall measure of surprise of the test data should decrease, since it is more likely that similar data was encountered. To restate, the intuition is this, the more experience our algorithm has seeing random walk data, the less surprising our particular section of random walk X_{RW} should appear.

We will not consider IMM and TSA-tree in this experiment. IMM is not defined for arbitrary large random walk datasets (see Section 7), and TSA-tree does not learn from experience, and thus will have a constant level of surprise. We tested TARZAN with increasing

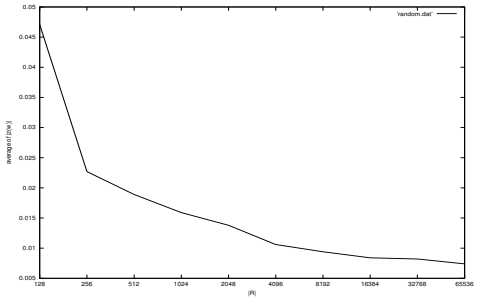

Figure 7: The average of $|z(w)|$ of all patterns w found by Tarzan in a small fixed random walk database X_{RW} when trained on increasing large random walk datasets. With more experience, Tarzan finds the patterns less surprising

large reference datasets length 128 to 65,536. We used a feature window length $l_1 = 12$ and an alphabet size $a = 4$.

The results are shown in Figure 7. Note that with little training data, the average values of $|z(w)|$ is quite high. In other words the patterns in X_{RW} appear surprisingly rare simply because TARZAN has not seen enough training data to build a general enough model of what to expect when confronted with more random walk data. However, with more experience, TARZAN finds the patterns in X_{RW} to be less and less surprising. These results strongly suggest that TARZAN is able improve its selectivity as it sees more data.

7. RELATED WORK

The task of finding surprising patterns in data has been an area of active research, which has long attracted the attention of researchers in biology, physics, astronomy and statistics, in addition to the more recent work by the data mining community. The problem, and closely related tasks are variously referred to as the detection of "Aberrant Behavior" [6, 24], "Novelties" [9, 5], "Anomalies" [37, 11], "Faults" [38], "Surprises" [32, 7], "Deviants" [20], "Temporal Change" [4, 14], and "Outliers" [18].

The problem of detecting surprising patterns in discrete domains has received the most attention, especially in the context of network intrusion [39]. However this problem differs from the task discussed in this paper in that the data is already discrete, it arrives at arbitrary intervals, and typically scalability to large datasets is not an important issue. In addition, while the data may be correlated, they are atomic units (for example, Unix commands, or ATM transactions), whereas time series data are intrinsically continuous. Various approaches to this problem have been suggested; many of them reduce to the idea that surprising patterns should be less compressible than unsurprising ones [7].

In the context of the analysis of biosequences, the search

for unexpectedly frequent or infrequent substrings is only one component of the broader quest for interesting patterns of more general kinds. Along these lines, patterns and families thereof have been variously characterized, and criteria, algorithms and software developed in correspondence. Without pretending to be exhaustive, we mention TEIRESIAS [31], GIBBS SAMPLER [25], WINNOWER, MEME [3], WINNOWER [29], PROJECTION [33], VERBUMCULUS [2, 1, 26], among others.

Detecting surprising patterns in time series has received much less attention. There has been some work on novelty detection in time series using neural networks [5, 37], however we do not consider this approach in detail since scalability is clearly an issue.

A simple approach to monitoring time series is to place a restriction on the maximum and minimum tolerable values, and to sound an alarm it the signal ever moves out of this envelope of acceptable behavior. This practice is referred to *limit-checking*. While trivial to implement, the method is known to have poor sensitivity. A more sophisticated approach involves *discrepancy-checking*. The idea here is to use the data observed thus far to predict future values. Any discrepancy between the two that exceeds a certain tolerance can be denoted as surprising [6]. Of course, this technique is only as good as the prediction algorithm, and time series prediction is a notoriously difficult problem. Thus, this method tends to have poor selectivity, producing many false alarms. In interesting work by Decoste [11], the author integrated both *limit-checking* and *discrepancy-checking* in a single framework, however scalability to massive datasets remains an issue, and the measure of surprise of an alarm can only be judged relative to the prediction model, which must be correctly specified by a domain expert.

Jagadish *et al.* [20] introduced a technique for mining deviants in time series, however deviants are simply *"...points with values that differ greatly from that of surrounding points"*, and thus this work may be considered more of a generalization of classic outlier detection [18].

In [32] and several follow up papers, Shahabi *et al.* suggest a method to find both trends and "surprises" in large time series datasets. The authors achieve this using a wavelet-based tree structure (TSA-Tree) that can represent the data at different scales, e.g., the weather trend in last month vs. last decade. However the definition of surprise used seems limited to dramatic shifts in the signal. In particular, this approach is not suitable for detecting unusual data patterns that hide inside the normal signal range. For example, the system would not be able to detect if we give it an EEG time series that we had flipped upside down, since the wavelet-based "surprise" features are invariant to this transformation of the data.

The immunological based approach of Dasgupta and Forrest [9], is inspired by the negative selection mechanism of the immune system, which discriminates between *self* and *non-self*. In this case *self* is the model of the time series learned from the reference dataset, and *non-self* are any observed patterns in the new dataset that do not conform to the model within some tolerance. A major limitation of the approach is that it is only defined when the space of *self* is not exhaustive. However, if you examine enough random walk data (or financial data, which is closely modeled by random walk [12]), *self* rapidly becomes saturated with every possible pattern, and thus *non-self* is the null set, and nothing encountered thereafter is considered surprising.

8. CONCLUSIONS AND FUTURE WORK

In this paper we introduced TARZAN, an algorithm that detects surprising patterns in a time series database in linear space and time. Our definition of surprising is general and domain independent, describing a pattern as surprising if the frequency with which we encounter it differs greatly from that expected given previous experience. We compared it to two other algorithms on both real and synthetic data, and found it to have much higher sensitivity and selectivity.

Although we see TARZAN's ability to find surprising patterns without user intervention as a great advantage, we intend to investigate the possibility of incorporating user feedback and domain based constraints.

For simplicity this work only considered one feature extraction technique, based on local slopes. However, it would be interesting to investigate other techniques such as wavelets and Fourier descriptors, and attempt to characterize which technique works best on which dataset.

Finally, because of space limitations we have concentrated solely on the intricacies of finding the surprising patterns, without addressing the many meta-questions that arise. For example, the possible asymmetric costs of false alarms and false dismissals, and the actionability of discovered knowledge [14]. We intend to address these issues in future work.

9. ACKNOWLEDGMENTS

We thank the anonymous referees for very useful comments on the paper.

10. REFERENCES

[1] A. Apostolico, M. E. Bock, and S. Lonardi. Monotony of surprise and large-scale quest for unusual words (extended abstract). In G. Myers, S. Hannenhalli, S. Istrail, P. Pevzner, and M. Waterman, editors, *Proc. of Research in Computational Molecular Biology (RECOMB)*, Washington, DC, April 2002.

[2] A. Apostolico, M. E. Bock, S. Lonardi, and X. Xu. Efficient detection of unusual words. *J. Comput. Bio.*, 7(1/2):71–94, Jan. 2000.

[3] T. L. Bailey and C. Elkan. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning*, 21(1/2):51–80, 1995.

[4] H. Blockeel, J. Furnkranz, A. Prskawetz, and F. C. Billari. Detecting temporal change in event sequences: An application to demographic data. In *Proc. Principles of Data Mining and Knowledge Discovery*, pages 29–41, 2001.

[5] R. Borisyuk, M. Denham, F. Hoppensteadt, Y. Kazanovich, , and O. Vinogradova. An oscillatory neural network model of sparse distributed memory and novelty detection. *BioSystems*, 58:265–272, 2000.

[6] J. D. Brutlag. Aberrant behavior detection in time series for network service monitoring. In *Proc. of the 14th Systems Administration Conference*, pages 139–146, Berkeley, CA, Dec. 3–8 2000. The USENIX Association.

[7] S. Chakrabarti, S. Sarawagi, and B. Dom. Mining surprising patterns using temporal description length. In *Proc. 24th Int. Conf. Very Large Data Bases*, pages 606–617, 1998.

[8] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *Proc. of the 4th International Conference of Knowledge Discovery and Data Mining*, pages 16–22. AAAI Press, 1998.

[9] D. Dasgupta and S. Forrest. Novelty detection in time series data using ideas from immunology. In *Proc. of The International Conference on Intelligent Systems*, 1999.

[10] C. S. Daw, C. E. A. Finney, and E. R. Tracy. Symbolic analysis of experimental data. *Review of Scientific Instruments 2001*, Oct. 30–31 2001.

[11] D. DeCoste. Mining multivariate time-series sensor data to discover behavior envelopes. In *Proc. Knowledge Discovery and Data Mining*, pages 151–154, 1997.

[12] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(2):419–429, June 1994.

[13] M. Farach. Optimal suffix tree construction with large alphabets. In *Proc. 38th Annual Symposium on Foundations of Computer Science*, pages 137–143, Oct. 1997.

[14] T. Fawcett and F. Provost. Activity monitoring: Noticing interesting changes in behavior. In *Proc. Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 53–62, 1999.

[15] W. Feller. *An introduction to Probability Theory and its Applications*. Wiley, New York, 1968.

[16] X. Ge and P. Smyth. Deformable markov model templates for time-series pattern matching. In *Proceedinmgs of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 81–90, 2000.

[17] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[18] D. M. Hawkins. *Identification of Outliers, Monographs on Applied Probability & Statistics*. Chapman and Hall, London, 1980.

[19] Y.-W. Huang and P. Yu. Adaptive query processing for time-series data. In S. Chaudhuri and D. Madigan, editors, *Proc. Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 282–286. ACM Press, Aug. 15–18 1999.

[20] H. V. Jagadish, N. Koudas, and S. Muthukrishnan. Mining deviants in a time series database. In *Proc. 25th International Conference on Very Large Data Bases*, pages 102–113, 1999.

[21] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 30(2):151–162, June 2001.

[22] E. Keogh and M. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *Proc. 4th International Conference on Knowledge Discovery and Data Mining*, pages 239–241, 1998.

[23] T. Kohonen. Self-organization of very large document collections: State of the art. In *Proc. 8th International Conference on Artificial Neural Networks*, volume 1, pages 65–74. Springer, 1998.

[24] E. Kotsakis and A. Wolski. Maps: A method for identifying and predicting aberrant behaviour in time series. In *Proc. 14th Internat. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, 2001.

[25] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton. Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science*, 262:208–214, Oct. 1993.

[26] S. Lonardi. *Global Detectors of Unusual Words: Design, Implementation, and Applications to Pattern Discovery in Biosequences*. PhD thesis, Department of Computer Sciences, Purdue University, August 2001.

[27] E. M. McCreight. A space-economical suffix tree construction algorithm. *J. Assoc. Comput. Mach.*, 23(2):262–272, Apr. 1976.

[28] S. Park, W. W. Chu, J. Yoon, and C. Hsu. Efficient searches for similar subsequences of different lengths in sequence databases. In *Proc. International Conference on Data Engineering*, pages 23–32, 2000.

[29] P. A. Pevzner and S.-H. Sze. Combinatorial approaches to finding subtle signals in DNA sequences. In *Proc. of the International Conference on Intelligent Systems for Molecular Biology*, pages 269–278. AAAI press, Menlo Park, CA, 2000.

[30] G. Reinert, S. Schbath, and M. S. Waterman. Probabilistic and statistical properties of words: An overview. *J. Comput. Bio.*, 7:1–46, 2000.

[31] I. Rigoutsos and A. Floratos. Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm. *Bioinformatics*, 14(1):55–67, 1998.

[32] C. Shahabi, X. Tian, and W. Zhao. Tsa-tree: A wavelet-based approach to improve the efficiency of multi-level surprise and trend queries. In *Proc. 12th International Conference on Scientific and Statistical Database Management*, 2000.

[33] M. Tompa and J. Buhler. Finding motifs using random projections. In *Annual International Conference on Computational Molecular Biology*, pages 67–74, Montreal, Canada, Apr. 2001.

[34] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[35] J. J. van Wijk and E. R. van Selow. Cluster and calendar-based visualization of time series data. In *Proc. IEEE Symposium on Information Visualization*, pages 4–9, Oct. 25–26 1999.

[36] P. Weiner. Linear pattern matching algorithm. In *Proc. 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11, Washington, DC, 1973.

[37] B. Whitehead and W. A. Hoyt. A function approximation approach to anomaly detection in propulsion system test data. In *Proc. AIAA/SAE/ASME/ASEE 29th Joint Propulsion Conference*, Monterey, CA, June 1993.

[38] T. Yairi, Y. Kato, and K. Hori. Fault detection by mining association rules from house-keeping data. In *Proc. of International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2001.

[39] J. Yang, W. Wang, and P. Yu. Info-miner: Mining surprising periodic patterns. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 395–400, 2001.