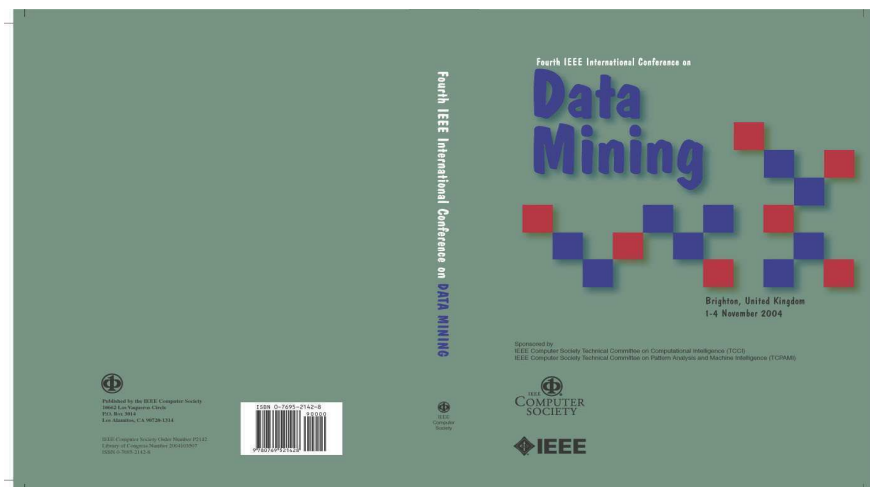


# Workshop Proceedings

## Temporal Data Mining: Algorithms, Theory and Applications

Held in conjunction with  
The Fourth IEEE International Conference on Data Mining (ICDM'04)



Organized by

Sheng Ma  
Tao Li  
Chang-shing Perng

November 01, 2004 Brighton, UK

# Preface

## 1. Temporal Data Mining

Many real-world applications deal with huge amounts of temporal data. Examples include alarms/events and performance measurements generated by distributed computer systems and by telecommunication networks, the web server logs, online transaction logs, financial data, and sensor data collected from sensor networks. Conventionally, temporal data is classified to either categorical event streams or numerical time series and both types have been intensively studied in data mining and statistics. However, several previously less emphasized aspects of temporal data have proven their importance in emerging applications and posed several challenges calling for more research.

The setting of traditional temporal data analysis is to apply one algorithm on a static, regular and relatively small temporal data set. Many practitioners found existing analysis methods inadequate for their real-world data. Many struggle to transform the data in order to apply existing methods or even to reduce the original problems to better studied ones; either ways induce in more preprocessing effort, more artificial parameters and less interpretable results. We believe these new aspects of temporal data deserve theories and algorithms of their own. Some of these new aspects are:

- Irregularity—Many types of numerical temporal data are not equally paced.
- Asynchronusness— In distributed computing environments like sensor networks, data from different sources tend to be not aligned and hence can not apply synchronous methods.
- Distributed analysis--- A trend in temporal data analysis is to perform data filtering, transformation and analysis as close as possible to the data sources to avoid the prohibitive amount of data being transmitted and analyzed. This new computing paradigm calls for a new theoretical foundation.
- Streaming data--- It is often desirable to analyze temporal data in (near-) real time for monitoring system health, detecting anomaly, alerting exceptional conditions, diagnosing problem, and performing actions including feedback control and self-healing of systems.
- Heterogeneous data types --- It is very common that temporal data is partly categorical events and partly numerical time series. It remains to be an interesting challenging to best analyze all possible data in a uniform way.
- Huge volume— the stream of data can be huge for a long, continuous observation period. Many types of measurements can be obtained from a large number of data sources. This requires designing scalable solutions in analyzing a large volume of temporal data, in terms of both the large number of data points and the large number of types of measurements.

Driven by the new aspects of temporal data, several fundamental problems need to be revisited. Just to name a few of them:

- Prediction
- Correlation

- Regression
- Benchmarking
- Periodic Pattern Mining
- Temporal Association Finding
- Sequential Event Patterns
- Causality Analysis
- Threshold selection
- Frequency Analysis
- Anomaly Detection
- Clustering
- Classification

## 2. The Workshop

The main objective of the “**Temporal Data Mining: Algorithms, Theory and Applications**” workshop is to bring together researchers from both industry and academia with different backgrounds: data mining, machine learning, database, statistical analysis, and application knowledge to propose new ideas, identify promising technologies and address the aforementioned technical challenges. The homepage for the workshop is <http://www.cs.rochester.edu/~taoli/workshop/>.

The workshop program included 8 technical papers selected by the program committee. It also included invited talks by Dr. Sheng Ma from IBM research and Dr. Oscar Kipersztok from Boeing Research.

We are very indebted to all program committee members who helped us organize the workshop and reviewed the papers very carefully. We would also like to thank all the authors who submitted their papers to the workshop; they provided us with an excellent workshop program.

November 2004

Workshop Co-chairs

Sheng Ma  
Tao Li  
Chang-shing Perng

# Workshop Agenda

**Date: Nov. 1, 2004 Afternoon**

**Location: Boardroom of The Old Ship Hotel, Brighton, England.**

Time	Authors	Titles
2:00--2:05	Workshop Chairs	Opening
2:05--2:45	Invited Speakers:  Dr. Oscar Kipersztok (Boeing Research) Dr. Sheng Ma (IBM Research)	Invited Talk:  Industrial Application of Temporal Data Mining
2:45--3:05	Tak-chung Fu, Fu-lai Chung, Robert Luk and Chak-man Ng (Hong Kong Polytechnic University Hong Kong Institute of Vocational Education)	<i>Financial Time Series Indexing based on Low Resolution Clustering</i>
3:05--3:25	Wei Xu, Peter Bodik and David Patterson (UC Berkeley)	<i>A Flexible Architecture for Statistical Learning and Data Mining from System Log Streams</i>
3:25--3:45	Ehud Gudes and Litvak Marina (Ben-Gurion University of the Negev, Israel)	<i>Discovering Target Event Rules based on Time-consecutive Pattern Mining</i>
3:45--4:15		Break
4:15--4:35	Lior Cohen, Gil Avrahami and Mark Last (Ben-Gurion University of the Negev, Israel)	<i>Incremental Info-fuzzy Algorithms for Real Time Data Mining of Non-stationary Data Streams</i>
4:35--4:55	Shen-Shyang Ho and Harry Wechsler (George Mason University)	<i>Learning from data streams via online transduction</i>
4:55--5:15	Jarkko Tikka and Jaakko Hollmen (Helsinki University of Technology)	<i>Learning linear dependency trees from multivariate time-series data</i>
5:15--5:35	Andreas D. Lattner and Otthein Herzog (University of Bremen, Germany)	<i>Unsupervised Learning of Sequential Patterns</i>
5:35--5:55	Francisco Guil, Alfonso Bosch and Roque Marin (University of Almeria)	<i>TSET<sup>MAX</sup>: An Algorithm for Mining Frequent Maximal Temporal Patterns</i>
5:55--6:00	Workshop Chairs	Closing

# Financial Time Series Indexing Based on Low Resolution Clustering

Tak-chung Fu<sup>1,2,‡</sup>, Fu-lai Chung<sup>1</sup>, Robert Luk<sup>1</sup> and Chak-man Ng<sup>2</sup>

<sup>1</sup>Department of Computing  
Hong Kong Polytechnic University  
Hungghom, Kowloon, Hong Kong.  
{cstcfu, cskchung, csrluk}@comp.polyu.edu.hk

<sup>2</sup>Department of Computing and Information Management  
Hong Kong Institute of Vocational Education (Chai Wan)  
30, Shing Tai Road, Chai Wan, Hong Kong.  
cmng@vtc.edu.hk

## Abstract

*One of the major tasks in time series database application is time series query. Time series data is always exist in large data size and high dimensionality. However, different from traditional data, it is impossible to index the time series in traditional database system. Moreover, time series with different lengths always coexists in the same database. Therefore, development of a time series indexing approach is of fundamental importance for maintaining an acceptable speed for time series query. By identifying the perceptually important points (PIPs) from the time domain, time series of different lengths can be compared and the dimensionality of the time series can be greatly reduced. In this paper, a time series indexing approach, based on clustering the time series data in low resolution, is proposed. This approach is customized for stock time series to cater for its unique behaviors. It follows the time domain approach to carry out the indexing process which is intuitive to ordinary data analysts. One may find it particularly attractive in applications like stock data analysis. The proposed approach is efficient and effective as well. As demonstrated by the experiments, the proposed approach speeds up the time series query process while it also guarantees no false dismissals. In addition, the proposed approach can handle the problem of updating new entries to the database without any difficulty.*

## 1. Introduction

Recently, time series data is of growing importance in many new database applications, such as data mining or data warehousing. A time series is a sequence of real numbers, each number representing a value at a time point. For example, the sequence could represent sales, exchange rates, weather data, biomedical measurements, stock prices ... etc.

We are often interested in finding similar time series [1] and querying time series database [2]. A time series database stores a large number of time series. The way of locating time series of interest in massive time series data efficiently is an important and non-trivial problem. Thus, indexing those time series is one of the possible solutions for increasing or at least maintaining an acceptable speed.

In this paper, an approach for indexing stock time series is proposed. A new time series representation scheme for various stock data analyses and mining tasks is first introduced. Stock time series has its own characteristics over other time series data such as the data from scientific areas like electrocardiogram ECG. For example, a stock time series is typically characterized by a few critical points and multi-resolution consideration is always necessary for long-term and short-term analyses. In addition, technical analysis is usually used to identify patterns of market behavior, which have high probability to repeat themselves. These patterns are similar in the overall shape but with different amplitude and/or duration. The perceptually important point (PIP) identification technique is adopted to determine data point importance and transform the original time series data into lower resolution. In addition, the time series database is indexed by clustering technique based on the low resolution of the time series. The paper is organized into six sections. Next section contains a discussion of related works. In section 3, the PIP time series representation framework based on data point reordering is introduced. The proposed time series indexing approach based on clustering is in Section 4. The simulation results are reported in Section 5 and the final section makes the conclusions of the paper.

## 2. Related Work

Reference [2] proposed the problem of similarity search in time series database and this problem has become an active research area afterward. Moreover, the problem of

---

<sup>‡</sup> Corresponding author

indexing time series has attracted many researchers in the database community. The original GEMINI framework with different dimensionality reduction approaches in reference [3] was applied to time series indexing. Furthermore, most of the techniques for signal analysis require the data which is in the frequency domain. Therefore, distance-preserving orthonormal transformations are often used to transform the data from the time domain to the frequency domain. Usually, data-independent transformation is applied to where the transformation matrix is determined by prior and independent input data. One of the most popular data-independent transformations is the Discrete Fourier transform (DFT) [2, 4, 5].

In reference [6], it identifies a safe set of Fourier transformations and presents a query processing algorithm that uses the underlying multidimensional index built over the data set to answer similarity queries efficiently. On the other hand, the similarity search methods perform dimensionality reduction on the data, and then use a multidimensional structure to index the data in the transformed space. The technique was introduced in reference [2] and extended in references [4, 5]. Since the distance between two signals in the time domain is the same as their Euclidean distance in the frequency domain, the DFT performs well in preserving essentials in the first few coefficients. For efficient access, a multidimensional index is constructed using the first few Fourier coefficients. When a similarity query is submitted, the index is used to retrieve the sequences that are at a certain small distance away from the query sequence.

Moreover, there are several kinds of indexing technique based on the frequency domain: Wavelets, Discrete Wavelet Transform (DWT) [7, 8, 9, 10], Singular Value Decomposition (SVD) [11] and Inner Products [12]. It can be concluded that traditional query process for time series data transforms data from the time domain into the frequency domain. However, it is less controllable by the end users and also affects the visualization ability. Therefore, this is not well suited for pattern queries that find time series patterns with many dynamical and specified user constraints such as stock time series queries.

On the other hand, the time domain approach works on the temporal data points directly, for example, Piecewise Aggregate Approximation (PAA) [13] and Adaptive Piecewise Constant Approximation (APCA) [14]. The majority of works focused solely on performance issues in the time domain, though some authors also considered other issues such as supporting non Euclidean distance measures [5, 13, 14] and allowing queries time series of different lengths [13, 14, 15]. There is another popular dissimilarity metric called the “time warping” distance. From the indexing viewpoint, this metric presents two major challenges: (a) it does not lead to any natural indexing “features”, and (b) it requires time quadratic in the sequence

length when comparing two sequences. To address each problem, reference [16] proposes to map sequences into points, with little compromise of “recall” for speeding up the query process. Moreover, reference [17] introduces a technique for the exact indexing of dynamic time warping. The same author gives a comprehensive survey on time series data mining in paper [18].

Furthermore, reference [15] presents a method to search time series data which first transforms time sequences into symbol strings using changeable ratio between contiguous data points in time series. Next, a suffix tree is built to index all suffixes of the symbol strings. The focus of this paper is to demonstrate how this method can adapt to the processing of the dynamically constrained time series queries. Recent work in reference [19] proposed a new symbolic representation and the transformed time series adopts the same indexing technique as reference [15].

There are much more time series indexing techniques [20, 21, 22, 23, 24] which address the problem of similarity search in large time series databases efficiently. To sum up, much of the recent database technology research on this problem, or similar ones, can be characterized procedurally in the following general manner [25]: (i) To find an approximation and robust representation for a time series, for example, Fourier coefficients, piecewise linear models, etc.; (ii) To define a flexible indexing function that can handle various pattern deformations (scaling, transformations, etc.); and (iii) To provide an efficient scalable and updatable algorithm by using the adopted representation and matching function for massive time series data sets. In this paper we proposed a time series indexing framework based on data point importance for dimensional reduction. It is found especially useful when focusing on time series domain that the fluctuations of the time series are important signals and multi-resolution consideration is necessary (i.e. technical analysis in financial domain).

### 3. Reordering of Time Series Data Points

In this section, the proposed time series representation scheme is introduced. It is based on the reordering of the data points in the time series. Instead of storing the time series data according to time, or transforming it into other domains (e.g. frequency domain), data points of a time series are stored according to their importance. Based on this representation, the time series can transform to low resolution for indexing. Two main steps are needed for reordering the time series: perceptually important point identification and data point importance list construction. They are described in the following subsections. Table I shows most of the notations used in this paper.

Table I. Notations used in this paper

Symbol	Meaning
$P$	A time series $P=p_1, \dots, p_m$
$Q$	A query pattern $Q=q_1, \dots, q_n$
$SP$	Low resolution from $P$ , $SP=sp_1, \dots, sp_n$ where $n \leq m$
$L$	The data point importance list $L=l_1, \dots, l_m$
$k$	Number of cluster
$M_i$	Cluster centroid $M=m_1, \dots, m_n$ where $i=1$ to $k$
$N$	Number of object in the database, $x_1, \dots, x_N$

### 3.1 Perceptually Important Point Identification

In view of the importance of extreme points in stock time series, the identification of perceptually important points (PIP) is first introduced in [26] and used for pattern matching of technical (analysis) patterns in financial applications. The idea was later found similar to a technique proposed about 30 years ago for reducing the number of points required to represent a line [27] (see also [28]). We also found independent works in [29, 30, 31] which work on similar ideas. In this subsection, let us review this idea by revisiting our previously proposed PIP identification process.

The frequently used stock patterns are typically characterized by a few critical points. For example, the head-and-shoulder pattern consists of a head point, two shoulder points and a pair of neck points. These points are perceptually important in the human identification process and should be considered of higher importance (Fig.1).

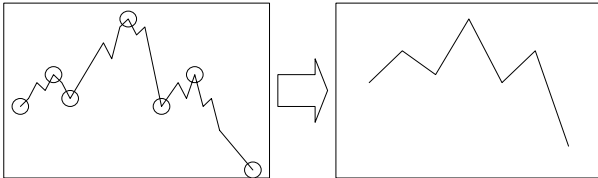


Figure 1. PIP identification results (from 30 data points in the data sequence  $P$  to 7 PIPs)

The PIP identification scheme follows this idea by locating those perceptually important points in the data sequence  $P$  in accordance with the query sequence  $Q$  (for pattern matching applications) or the specified number of PIPs. The locating process works as follows.

With sequences  $P$  and  $Q$  being normalized (for shifting and uniform amplitude scaling invariant), the PIPs are located in order according to Fig.2. Currently, the first two PIPs will be the first and the last points of  $P$ . The next PIP will be the point in  $P$  with maximum distance to the first two PIPs. The fourth PIP will then be the point in  $P$  with maximum distance to its two adjacent PIPs, either in between the first and second PIPs or in between the second and the last PIPs. The PIP location process continues until its number is equal to the length of query sequence  $Q$  or the required number of

PIPs is reached.

```

Function Point_locate (P,Q)
  Input:  sequence P[1..m], length of Q[1..n]
  Output: pattern SP[1..n]
Begin
  Set sp[1]=p[1], sp[n]=p[m]
  Repeat until SP[1..n] all filled
  Begin
    Select point p[j] with maximum distance to
    the adjacent points in SP (sp[1] and sp[n]
    initially)
    Add p[j] TO SP
  End
  Return SP
End
  
```

Figure 2. Pseudo code of the PIP identification process

The distance metric for distance measurement (i.e. calculation of fluctuation value of a data point) is the vertical distance (VD) between a test point  $p_3$  and the line connecting the two adjacent PIPs, i.e.,

$$VD(p_3, p_c) = |y_c - y_3| = \left| \left( y_1 + (y_2 - y_1) \cdot \frac{x_c - x_1}{x_2 - x_1} \right) - y_3 \right| \quad (1)$$

where  $x_c = x_3$ . Eqn. (1) is intended to capture the fluctuation of the data sequence and the highly fluctuated points (or locally extreme points) would be considered as PIPs.

To illustrate the identification process, Fig.3 shows the step-by-step results. Here, the number of data points in the input sequence  $P$  and query sequence  $Q$  (or the required number of PIPs) are 10 and 5 respectively, i.e.,  $m=10$  and  $n=5$ .

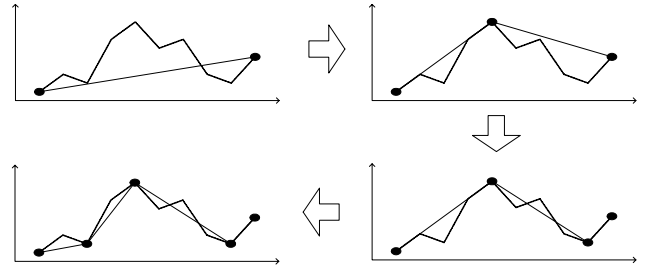


Figure 3. Identification of 5 perceptually important points

### 3.2 Building Data Point Importance List

Inspired from the above idea, a reordering mechanism for the data points in a time series is proposed in paper [32]. Instead of only identifying  $n$  PIPs from the longer sequence  $m$  for pattern matching, only one time series is provided in the current scenario. Therefore, all the data points in the time series  $P$  will go through the PIP identification process. In other words, the number of PIPs to be identified is equal to the number of data points in the given time series. However, the sequence of the data points being identified is the main focus.

The sequence of PIPs is ordered according to Fig.4. It is the same as the original process except the PIP identification process continues until all the data points in the given time series  $P$  have been processed and appended to the list  $L$ .

```

Function Build_list (P)
  Input: sequence P[1..m]
  Output: sequence L[1..m]
Begin
  Set l[1]=p[1], l[2]=p[m]
  Repeat until l[1..m] all filled
  Begin
    Select point p[j] with maximum distance to
    the adjacent points in the list (l[1] and
    l[2] initially)
    Append point p[j] TO L
  End
  Return L
End

```

Figure 4. Pseudo code of the data point importance list construction

According to Fig.4, the PIPs identified in the earlier iterations should be considered as more important than those points identified later. Therefore, they have higher importance. Continued with the example in the previous sub-section, Fig.5 shows the reordered sequence of the data points based on the PIP identification process. The points with smaller number label are more important (e.g. PIP 3 is more important than PIP 4). Table II shows the data point importance list built for this example. Besides the importance of the PIPs, their corresponding amplitude  $y$  and vertical distance measured are also stored in the table for future references. A point needed to be mentioned here is that the fluctuation value, i.e. the VD value, is not guaranteed to decrease with the decrease in importance. It is because the measurement of VD depends on the global shape captured in previous PIP identification iteration but not an overall shape. So, the change of such shape will affect the calculation of VD of the data points in the original time series compared with the pattern formed by the identified PIPs (e.g. PIPs with importance 6 and 7).

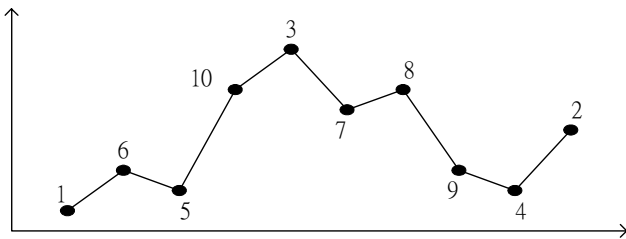


Figure 5. The order of points found by the PIP identification process

Table II. Example of data point importance list

Importance	$x$	$y$ (Amplitude)	Fluctuation (VD)
1	1	0.1	n/a
2	10	0.5	n/a
3	5	0.9	0.62
4	9	0.3	0.28
5	3	0.3	0.20
6	2	0.4	0.20
7	6	0.6	0.15
8	7	0.7	0.20
9	8	0.4	0.10
10	4	0.7	0.10

With the construction of the data point importance list, the retrieval, analysis and processing of time series can be carried based on this list and different degrees of benefit, in terms of efficiency and effectiveness, can be obtained. A tree structure representation for storing the list is proposed in paper [32]. In this paper, the data point importance list is applied for dimensionality reduction to represent the time series in lower resolution for indexing.

#### 4. Stock Time Series Indexing based on Clustering

To facilitate time series indexing, representing the time series in low dimension is of fundamental importance. It is because time series data with different lengths always coexists and measuring the distance among them is the necessary step of the indexing process. Furthermore, a mechanism for representing time series in a low resolution can improve the efficiency of the indexing algorithm and maintain the visualization ability. Reference [33] further shows that initializing the clusters' centroids on a low resolution of the data can improve the quality. Therefore, the overall process for the proposed stock time series indexing scheme can be classified as two main components: dimensionality reduction of the time series data to a lower resolution and the clustering process for time series indexing.

##### 4.1 Dimensionality Reduction by Accessing the Data Point Important List

With the data point importance list  $L$  built for the time series  $P$ , the low resolution of the time series  $SP$  can be obtained by selecting  $n$  number of PIPs from the top of the list as Fig.6, where  $n \ll m$ . By applying this method, all the time series in the database can provide a lower resolution version of representation with uniform length for indexing and, therefore, dimensionality reduction can be achieved.

```

Function PIP_retrieval (L, n)
  Input:  sequence L[1..m]
  Output: sequence SP[1..n]
Begin
  For i = 1 to n
  Begin
    Append point l[i] TO SP
  End
  Return SP
End

```

Figure 6. Data point retrieval from the data point important list for dimensionality reduction

Referring to the example in section 3.2, if  $n=3$ , it means that the first 3 PIPs, (10, 0.5), (1, 0.1) and (5, 0.9), should be retrieved from the data point important list to represent the time series in lower resolution. The visualization result of the low resolution of the sample time series is as Fig.7.

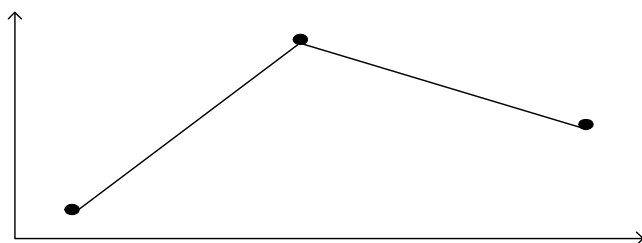


Figure 7. Low resolution ( $n=3$ ) of the sample time series in Fig.5

#### 4.2 Building Index by Clustering

After we transformed all the time series in the database to a lower resolution, we can group the time series with similar shape together for indexing. Clustering is a common method for finding structure in given data [9, 11], in particular for finding structure related to time. There are many popular clustering techniques developed, such as hierarchical, nearest neighbor, and  $k$ -means algorithms. In the data mining context, the most common one perhaps is  $k$ -means algorithm [34] and it is good enough for our time series indexing process.

In  $k$ -means clustering, each cluster is represented by the center of the cluster, centroid. The  $k$ -means algorithm is implemented in 4 steps: (1) cluster objects into  $k$  nonempty subsets, (2) compute seed points as the centroids of the clusters of the current partition, (3) Assign each object to the cluster with the nearest seed point and (4) go back to the step 2, stop when no more new assignment or excess the maximum number of iteration.

Suppose that there exist  $N$  objects,  $x_1, x_2, \dots, x_N$ , in the database, and we know that they fall into  $k$  compact clusters,  $k \ll N$ . Let  $M_i$  be the mean of the vectors, that is, the cluster centroid, in cluster  $i$ . If the clusters are well separated, a minimum distance classifier can be used to separate them. That is, we can say that  $x$  is in cluster  $i$  if  $|x - M_i|$  is the

minimum of all the  $k$  clusters. This suggests the procedure in Fig.8 for finding the  $k$  means.

```

Function k_means_clustering (N)
  Input:  object x[1..N]
Begin
  Make initial guesses for the means  $M_1, M_2, \dots, M_k$ 
  Until there are no changes in any mean or
  excess the maximum number of iteration
  Use the estimated means to classify the
  objects into clusters
  For i from 1 to k
  Replace  $M_i$  with the mean of all the
  objects for cluster  $i$ 
  End for
  End Until
End

```

Figure 8.  $k$ -means clustering process

To index the time series database by clustering process, all the time series in the database are the input objects for the clustering process and the low resolution of these time series is considered as the input feature vectors. As the low resolution of the time series  $n$  is smaller or equal to the smallest length among all time series in the database, all the time series can be indexed even though the length of original time series in the database are different. Finally, each of the time series will group into one cluster among  $k$  clusters as Fig.9.

The distance between the low resolution of the time series ( $SP$ ) from time series ( $P$ ) and the cluster centroid  $M_i$  can be computed using direct point-to-point comparison, that is,

$$Dist(SP, M_i) = \sqrt{\frac{1}{n} \sum_{j=1}^n (sp_j - m_{j,k})^2} \quad (2)$$

for all centroids,  $M$ , during the clustering process.

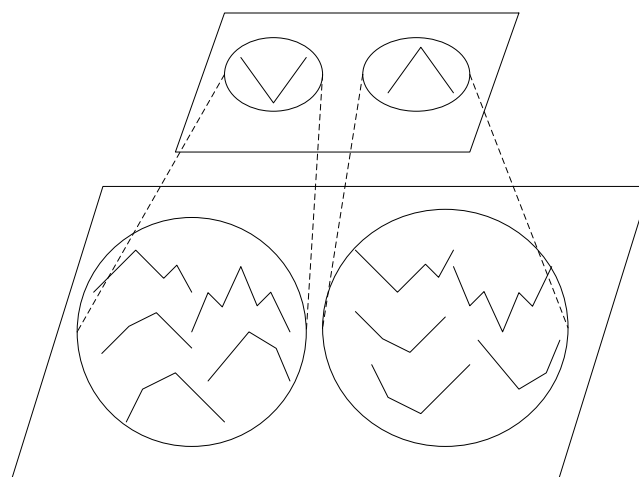


Figure 9. Time series clustering

### 4.3 Updating the Index

In addition, it is an easy task to update the index when adding a new time series  $P_{new}$ . The data point importance list of the new time series should first be built and the resolution of this time series can then be reduced to  $n$ ,  $SP_{new}$ . Afterwards, it can be added to the cluster  $i$  that  $|SP_{new}-M_i|$  is the minimum of all the  $k$  clusters, or else, a new cluster has to be created for this new time series if  $|SP_{new}-M_i|>\lambda$ , where  $\lambda$  is an user defined threshold (Fig.10).

```

Function Incremental_update (Pnew)
  Input: sequence Pnew[1..m]
Begin
  Lnew = Build_List (Pnew)
  SPnew = PIP_retrieval (Lnew)

  Find min(|SPnew-Mi|) for all cluster (M1 to Mk)

  If min(|SPnew-Mi|)>λ
    Create a new cluster Mk+1 for Pnew
  Else
    Append Pnew to cluster i
  End If
End

```

Figure 10. Incremental updating of the index

### 4.4 Query the Index

After building the index, the users can query the time series database by first providing the shape of the time series,  $Q$  (i.e. query by example) with  $n$  number of data points or reducing the dimension of  $Q$  to  $n$  by building the proposed data point important list, where  $n$  is same as the low resolution of the time series in the database. Direct comparison can be applied between  $Q$  and the cluster centroids,  $M$ , using Eq.2. By identifying which cluster the query sequence belongs to (with minimum distance), only the time series which belongs to this cluster is needed to evaluate. By this method, the number of time series needed to examine can be reduced while the overall shape of the time series can still be preserved. Fig.11 shows the query procedure.

```

Function Query (Q)
  Input: sequence Q[1..n]
  Output: sequence P[i..m]
Begin
  Find min(|Q-Mi|) for all cluster (M1 to Mk)
  Examine all the time series belongs to Mi and
  find P which with the minimum distance with Q
  Return P
End

```

Figure 11. Query process for our indexing approach

## 5. Simulation Results

In this section, the performance of the proposed time series indexing algorithm will be reported. The dataset used was taken from the opening price time series of the Hong Kong stock market. 200 stock time series with the length between 251 and 1685 were considered. And the average

length is 1537. Five different time series were randomly selected from the dataset to serve as the query series and the figures of all the experiments were the average of the five results.

Firstly, the effects on space consumption when increasing the resolution, which are the number of PIP retrieved  $n$  and the number of cluster  $k$ , were tested. Obviously, more space was required when increasing either the number of PIP or the number of cluster (Fig.12).

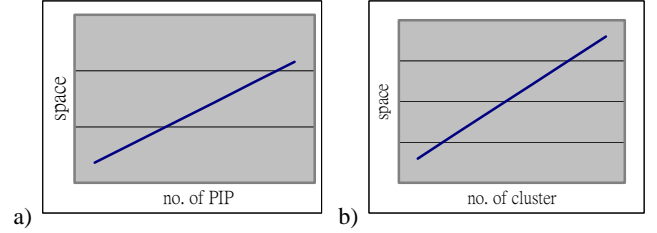


Figure 12. (a) Space consumption vs. number of PIP and (b) Space consumption vs. number of cluster

Secondly, the CPU cost (i.e. speed) and the number of iteration needed to converge for building the index with different parameters settings, which are the resolution (number of PIP retrieved,  $n$ ) and the number of cluster  $k$ , were tested. The number of clustering is fixed to 4. As shown in Fig.13, the increase of number of PIP raised the time for building the index. It is because the increase of number of PIP augments the dimension of the input feature vector for the clustering process. Moreover, it was more difficult for the clustering process to converge in a higher dimension. More iterations were also needed (Fig.14). When the number of PIP was set to 15 and 20, the maximum number of iteration (i.e. 500 cycles) was reached. It means that the index built by these two experiments might have false dismissals as the clusters were not well separated. Fig.15 shows the shapes of the cluster centroids after the clustering process with different numbers of PIP. Similar shapes were obtained in different resolutions but more detail of the shapes was shown in higher resolution.

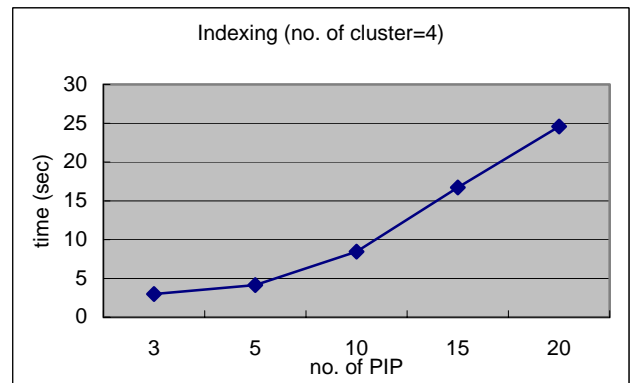


Figure 13. CPU cost for building the index against number of PIP ( $n$ )

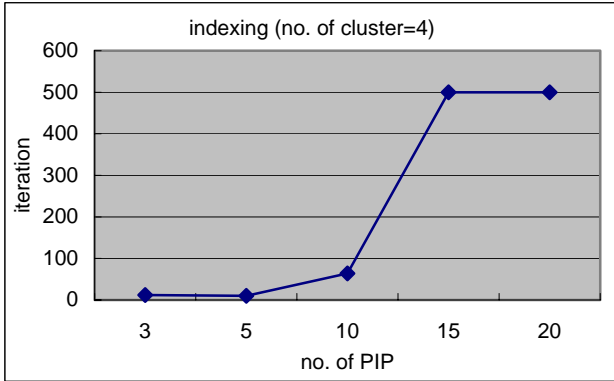


Figure 14. Number of iteration for building the index against number of PIP (n)

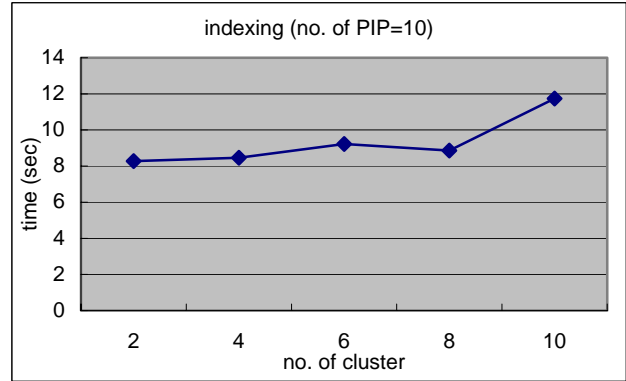


Figure 16. CPU cost for building the index against number of cluster (k)

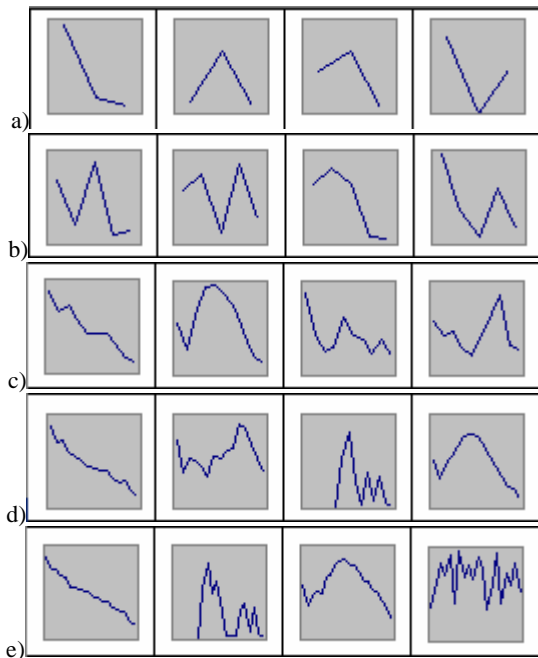


Figure 15. Visualization of clusters' centroids (a) n=3 (b) n=5 (c) n=10, (d) n=15 and (e) n=20

Then, the number of PIP was fixed to 10. When increasing the number of cluster, the time for building the index increased more stably compared to the increase of the number of PIP. The result of the experiment shows that the dimension of the input feature vector dominates the time for the indexing process when comparing to the number of clustering. Furthermore, Fig.17 shows that the dependency between the number of cluster and the number of iteration needed for building the index was low but it should be in direct proportion.

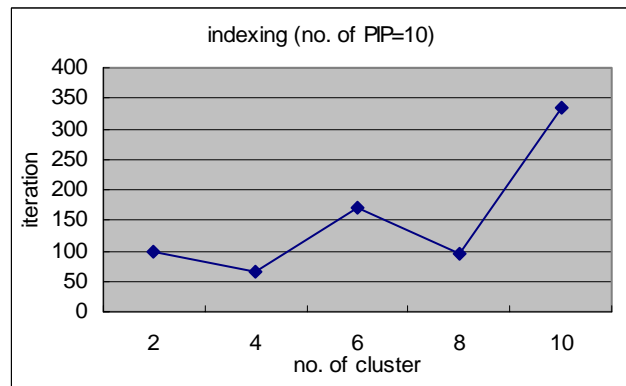


Figure 17. Number of iteration for building the index against number of cluster (k)

Thirdly, the pruning power and the CPU cost of the proposed algorithm were measured with different n and k during the query process. Pruning power is defined as Eq. (3) [14].

$$P = \frac{\text{Number of object that must be examined}}{\text{Number of object in the database}} \quad (3)$$

In this part, the effect of n was tested first. The number of cluster k was fixed to 4. As shown in Fig.18 and Fig.19, both the pruning power and the query time were independent to the number of PIP but rather dependent on the number of cluster. Then, n is fixed to 10, and the effect of using different number of cluster k was tested. As shown in Fig.20 and Fig.21, the increase of number of clustering improved the query performance. However, as the size of the dataset was not large, the improvement was kept in stable after 4 clusters were used.

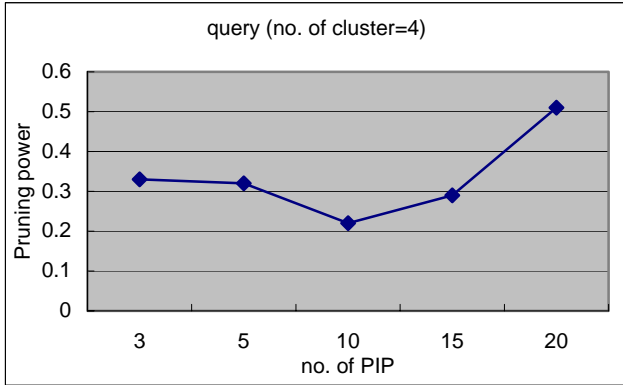


Figure 18. Pruning power against number of PIP ( $n$ )

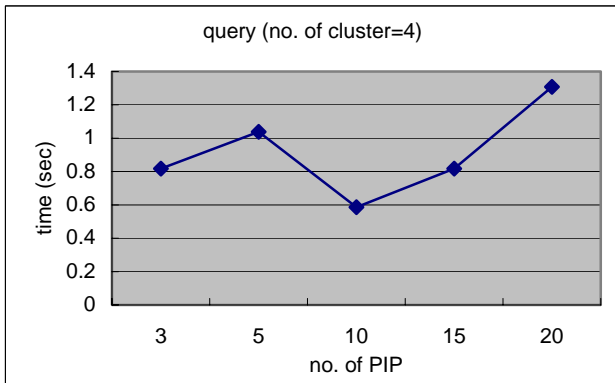


Figure 19. Query time against number of PIP ( $n$ )

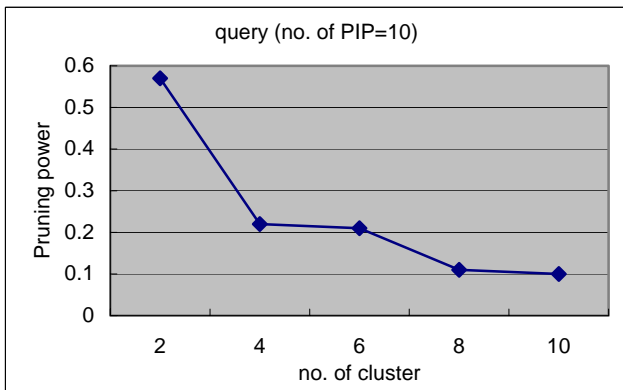


Figure 20. Pruning power against number of cluster ( $k$ )

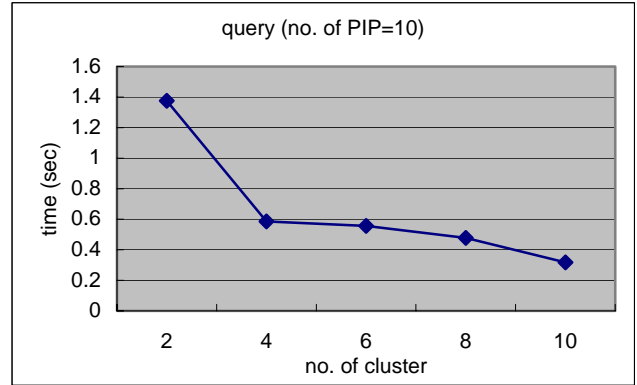


Figure 21. Query time against number of cluster ( $k$ )

In the above experiments, although time is needed to build the index of the time series database, this is only a one-time process. On the other hand, the query time can be reduced in an obvious degree with no false dismissals when the clustering process could be converged before reaching the maximum number of iteration. It can be easily achieved by considering the low resolution (the number of PIP  $n$ ) of the input feature vector. Moreover, the time series database can be updated easily as suggested in section 4.3. Finally, Fig.22 shows the comparison on the query time between the sequential scanning of the time series database and the proposed indexing approach ( $n=10$  and  $k=4$ ). Obviously, the proposed approach outperformed the sequential scanning and the query time could be kept stable when increasing the number of object in the database.

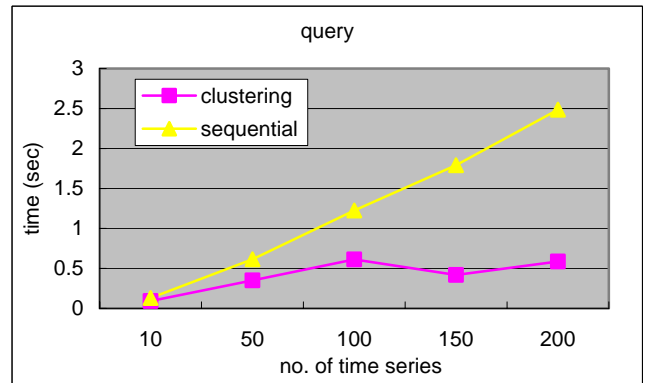


Figure 22. Comparison on the query time between sequential scanning and the proposed indexing approach

## 6. Conclusions

In this paper, a stock time series indexing approach is proposed. It is based on data point importance to reduce the dimension of the time series data. Then, the clustering process is applied to the low resolution of the time series for indexing. This method can solve the problem of the coexisting of time series with different lengths in the database. It is not only efficient but also effective. The proposed approach is customized for stock time series data

which has its unique behaviors. As demonstrated by the experiments, the proposed approach speeds up the time series query process while no false dismissals can be guaranteed. In addition, the proposed approach can handle the updating problem of the time series database without any difficulty. For further development, it is possible to further enhance the indexing process by applying the concept of hierarchical indexing. That is, multiple-resolution clustering technique can be applied and clustered the time series data from low resolution to high resolution.

## References

1. G. Das, D. Gunopulos and H. Mannila, "Finding Similar Time Series," *In proceedings of the 1<sup>st</sup> European Conference on Principles and Practice of Knowledge Discovery in Databases*, pp.88-100, 1997.
2. R. Agrawal, C. Faloutsos and A.N. Swami, "Efficient similarity search in sequence databases," *In proceedings of the 4<sup>th</sup> International Conference on Foundations of Data Organization and Algorithms*, pp.69-84, 1993.
3. Faloutsos, M. Ranganathan and Y. Manolopoulos, "Fast subsequence matching in time series databases," *In proceeding of the ACM SIGMOD International Conference on Management of Data, Minneapolis*, pp. 419-429, 1994.
4. K. Chu and M. Wong, "Fast time-series searching with scaling and shifting," *In proceedings of the 18<sup>th</sup> ACM Symposium on Principles of Database Systems*, pp. 237-248, 1999.
5. Refiei, "On similarity-based queries for time series data," *In proceedings of the 15<sup>th</sup> International Conference on Data Engineering*, pp. 410-417, 1999.
6. D. Rafiei and A.O. Mendelzon, "Querying Time Series Data Based on Similarity", *IEEE Transactions on Knowledge and Data Engineering*, 12 (5), pp.75-693, 2000.
7. T. Kahveci and A. Singh, "Variable length queries for time series data," *In proceedings of the 17<sup>th</sup> International Conference on Data Engineering*, pp. 273-282, 2001.
8. Popivanov and R. J. Miller, "Similarity search over time series data using wavelets," *In proceedings of the 18<sup>th</sup> International Conference on Data Engineering*, pp. 212-221, 2002.
9. C. Wang and X. S. Wang, "Supporting content-based searches on time series via approximation," *In proceedings of the 12<sup>th</sup> International Conference on Scientific and Statistical Database Management*, pp. 69-81, 2000.
10. Y. Wu, D. Agrawal and A. El Abbadi, "A comparison of DFT and DWT based similarity search in time-series databases," *In proceedings of the 9<sup>th</sup> ACM CIKM International Conference on Information and Knowledge Management*, pp. 488-495, 2000.
11. K.V. Kanth, D. Agrawal and A. Singh, "Dimensionality reduction for similarity searching in dynamic databases," *In proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 166-176, 1998.
12. Ferhatosmanoglu, E. Tuncel, D. Agrawal and A. El Abbadi, "Approximate nearest neighbor searching in multimedia databases," *In proceedings of the 17<sup>th</sup> IEEE International Conference on Data Engineering*, pp. 503-511, 2001.
13. B. Yi and C. Faloutsos, "Fast time sequence indexing for arbitrary Lp norms," *In proceedings of the 26<sup>th</sup> International Conference on Very Large Databases*, pp. 385-394, 2000.
14. E. Keogh, K. Chakrabarti, M. Pazzani and S. Mehrotra, "Locally adaptive dimensionality reduction for indexing large time series databases," *In proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 151-162, 2001.
15. Y. W. Huang and P. S. Yu, "Adaptive Query Processing for Time Series Data", *In proceedings of the 5<sup>th</sup> ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp 282-286, 1999.
16. B. Yi, H.V. Jagadish and C. Faloutsos, "Efficient Retrieval of Similar Time Sequences Under Time Warping," *In proceedings of the 14<sup>th</sup> International Conference on Data Engineering*, pp.201-208, 1998.
17. E. Keogh, "Exact Indexing of Dynamic Time Warping," *In proceedings of the 28<sup>th</sup> International Conference on Very Large Data Bases Conference*, pp.406-417, 2002.
18. E. Keogh and S. Kasetty, S, "On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration", *In proceedings of the 8<sup>th</sup> ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp 102-111, 2002.
19. J. Lin, E. Keogh, S. Lonardi and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," *In proceedings of the 8<sup>th</sup> ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2003.
20. E. Keogh and M. Pazzani, "An Indexing Scheme for Fast Similarity Search in Large Time Series Databases," *In proceedings of the 11<sup>th</sup> International Conference on Scientific and Statistical Database Management*, pp.56-67, 1999.
21. H.A. Henrik and D. Z. Badal, "Using Signature Files for Querying Time Series Data", *In proceedings of Principles of Data Mining and Knowledge Discovery*, pp 211-220, 1997.
22. T. Sellis, N. Roussopoulos, C. Faloutsos, "The R+-Tree: A Dynamic Index For Multi-Dimensional Objects", *The Very Large Database Journal*, 1987.
23. I. Zwir and E.H. Enrique, "Qualitative Object Description: Initial Reports of the Exploration of the Frontier," *In proceedings of the Joint EUROFUSE-SIC'99 International Conference*, 1999.
24. K. Bennett, U. Fayyad and D. Geiger, "Density-based indexing for approximate nearest-neighbor queries," *In proceedings of the 5<sup>th</sup> International Conference on Knowledge Discovery and Data Mining*, pp. 233-243, 1999.
25. X. Ge and P. Smyth, "Deformable Markov model templates for time-series pattern matching," *In proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.81-90, 2000.
26. F.L. Chung, T.C. Fu, R. Luk and V. Ng, "Flexible time series pattern matching based on perceptually important points," *International Joint Conference on Artificial Intelligence Workshop on Learning from Temporal and Spatial Data*, pp.1-7, 2001.
27. D. Douglas and T. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *The Canadian Cartographer*, vol.10, no.2, pp.112-122, 1973.
28. J. Hersherberger and J. Snoeyink, "Speeding up the Douglas-Peucker line-simplification algorithm," *In the proceedings of the 5<sup>th</sup> Symposium on Data Handling*, pp.134-143, 1992.

29. C.S. Perng, H. Wang, R. Zhang and D. Parker, "Landmarks: A new model for similarity-based pattern querying in time series databases," *In proceedings of the 16<sup>th</sup> International Conference on Data Engineering*, pp.33-42, 2000.
30. B. Pratt and E. Fink, "Search for patterns in compressed time series," *Image and Graphics*, vol.2, no.1, pp.89-106, 2002.
31. E. Fink and B. Pratt, "Indexing of compressed time series," *Data Mining in Time Series Databases*, pp.51-78, 2003.
32. T.C. Fu, F.L. Chung, R. Luk and C.M. Ng, "A Specialized Binary Tree for Financial Time Series Representation," *The 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining Workshop on Temporal Data Mining*, 2004.
33. C. Ding, X. F. He, H. Y. Zha and H. D. Simon, "Adaptive dimension reduction for clustering high dimensional data," *In proceedings of the 2<sup>nd</sup> IEEE International Conference on Data Mining*, pp. 147-154, 2002.
34. U. Fayyad, C. Reina and P. Bradley, "Initialization of Iterative Refinement Clustering Algorithms," *In proceedings of the 4<sup>th</sup> International Conference on Knowledge Discovery and Data*, pp. 194-198, 1998.

# A Flexible Architecture for Statistical Learning and Data Mining from System Log Streams

Wei Xu    Peter Bodík    David Patterson  
EECS Department, UC Berkeley  
{xuw, bodikp, patterson}@cs.berkeley.edu

## Abstract

*Modern computer systems are instrumented to generate huge amounts of system log data. This data contains valuable information for managing the system, localizing failures, and recovery. However, the complexity of these systems greatly surpasses what can be understood by human operators and thus automated analysis systems are beginning to be used. Due to preprocessing required by the statistical algorithms, the extremely high volume of data cannot be processed using ad-hoc scripts. We present a flexible, modular and scalable architecture for statistical learning from large data streams that can easily process lots of data. We built a prototype that is evaluated using system log data from a commercial on-line service. Moreover, the results of the analysis were genuinely useful for the on-line service operators.*

## 1. Introduction

### Data analysis and collection in general

Most on-line services such as Amazon or eBay suffer from various user-visible failures. As reported by various authors (such as [14]), the most common causes of failure in computer systems are *software bugs*, *human operator errors* and *hardware failures*. These failures cause system downtime which is very expensive; for example, [15] estimates that 1 hour of downtime of a large on-line service can cost up to \$1 million. Predicting, detecting or localizing these failures is a very difficult task – some systems consist of hundreds of software components running on up to 50,000 servers [13].

A standard approach of most companies is to instrument the system so that it reports various statistics such as performance of each machine, execution details for each request, or network statistics. Human operators monitoring the system use the log data and their experience to detect failures and identify possible root causes.

This approach has several limitations: computer systems are too complex and their behavior cannot be easily under-

stood by humans. The systems today generate as much as 1 TB of log data every day [4]. Using more fine-grained instrumentation they could generate 100x more log data, but it is impossible to manually process such large data sets.

### Motivation

The recorded data contains more useful information than the operators can discern. The data might be used to predict that a particular machine is likely to crash in a few minutes or that a certain software component has a bug. This extra information can expedite detection and localization of failures. A fully automated analysis would be cheaper, more reliable and could potentially understand more complex behaviors. However, most of the companies today use very limited automated analysis of the log data they collect.

One of the reasons why it is difficult to build an automated analysis system is that it is hard to manage so much data in real time. A lot of preprocessing (such as sampling, adding/removing attributes, merging data from different sources, and so on) is required before the data reach the algorithms. Our experience shows that systems based on Python scripts are not flexible enough to analyze the scale of log data produced in on-line services. Using ad-hoc scripts for parallel preprocessing of data is tedious and does not allow easy modification of data streams and algorithms.

Instead, we need a better data model for the system log, and a scalable, modular architecture that can be distributed over a cluster of machines to process the data quickly. We need to be able to easily add new data streams of different types and data rates, create new features/attributes on-line, and try different types of algorithms. Since we need to test the system in production environments, it needs to be easily deployable for testing.

### Contribution

In this paper we propose the use of stream-based processing and present a prototype of a system that allows us to preprocess data for any off-line or on-line statistical learning algorithm against massive quantities of system logs. The system can be easily distributed over a cluster of machines and new data streams and algorithms can be added on-the-fly.

Although our architecture is designed for analyzing system log data, it can be used in other situations where data mining and statistical learning theory (SLT) algorithms need to be applied to huge amounts of data. To demonstrate its benefits, we implemented a decision tree algorithm for generating interesting rules from the log data. The algorithms and the architecture are evaluated on a data set from an on-line service with data rate of about 150GB per day.

The structure of the paper is as follows: in Section 2 we present a set of decision-tree-based algorithms for off-line detection of interesting rules/behavior in a typical computer system; Section 3 highlights practical problems we ran into when trying to implement them. Sections 4 and 5 describe the new system and our experience with it. The results of the algorithms in Section 6 are followed by the details of the design and implementation of the system in Section 7.

## 2. System log data analysis

In this section we describe a typical data set obtained from an on-line service that we use in our experiments. We also describe a few types of algorithms that would be useful for system operators and present some specific examples. These algorithms and problems with their implementation serve as motivation for our stream-based system.

### 2.1. Data set X

Our data set comes from a commercial on-line service (similar to AOL) running on 440 nodes. Since failure information is sensitive, the company sharing this data prefers to be anonymous (we refer to it as company X or system X). The data was recorded during a 20-day period and its size is about 2.5 TB.

The data set contains three types of data:

**Request data.** Every request executed in the system reports at least the following attributes: *time*, *machine*, *user id*, and *application*. In addition to that, a request reports a subset of 450 attributes including *request type*, *content length* or *queue duration*; the actual attributes that are reported depend on the application that is executed and the events that happen during the execution. Note that *requests* are not restricted to requests from *users*; requests from other parts of the system are also included. Due to privacy reasons, some of the attributes were *anonymized*; replaced by a hash of their string value. The peak rate of requests is about 11,000 requests per second. Each 1 minute of request data from all machines is stored in one compressed file of size  $\approx 20$  MB (uncompressed size is  $\approx 80$  MB).

**Performance data.** Every node reports its performance statistics every five minutes – 17 attributes such as: *memory utilization*, *swap utilization*, *load average*, *CPU idle time*,

or *TCP segments received in error*. The size of the performance data for 20 days is  $\approx 850$  MB; the data rate is about 10,000 times slower than for the request data.

**Trouble ticket data.** The problems detected by operators were written to a log ( $\approx 450$  kB) that includes possible causes for the failures.

### 2.2. What is useful

Our experience with several companies suggests that the following types of automatic log data analysis would be useful in general:

1. **localizing the root cause of a failure:** Often the system operators know that *something* in the system failed, but localization of the root cause is a significant problem. As reported by [9], one large site estimates that about 93% of recovery time from application-level failures is spent in detecting and diagnosing them.
2. **predicting failures before they happen:** If we could predict failures of particular machines, we could redirect the traffic from the affected machine and avoid service disruptions.
3. **detecting (unexpected) patterns in the data:** During the *post-mortem* analysis of a failure it would be useful to have models of typical and unexpected system behavior. These models would allow the operators to better understand the system.

The automated analysis systems should not replace the operators but help them understand the system better. Human operators are experienced with the system, so we need to exploit it in our analysis. The analysis systems should thus also incorporate feedback from the operators.

### 2.3. Analyzing the data

It is still not known what algorithms work best for system log analysis. However, as summarized in Section 8, current research mostly applies off-line algorithms such as decision trees, Bayes nets or association rules along with time series analysis and statistical tests. For analysis of our data set we decided to start with a simple decision tree algorithm and then gradually extend it.

#### Decision trees for important attributes

The basic algorithm uses decision trees to generate interesting rules about the behavior of the system. Some of the reported attributes indicate a possible error: *error-code*, *response-code*, *result*, or *db-error*. For the system operators it is important to understand what requests report different values of such attributes. Since we prefer output in human-readable format, it is natural to use decision trees to classify

the values of such attributes using the values of the remaining attributes.

Thus, this simple algorithm would look at all requests on one machine during a specified time interval (say, an hour), treat one of the attributes as a *class attribute* and train a decision tree using this data set. An operator can then easily extract useful rules that define the classes.

It is often not enough to use only the original attributes from the raw data. An operator who understands the system might like to add *new attributes* that he would like to classify or use in classification of other attributes. For example, since most attributes are not reported most of the time (such as *error-code*), the presence of an attribute indicates an event of interest. Therefore, for every attribute  $A$  in the original raw log data, we define new boolean attribute  $R_A$ : "*is attribute A reported?*".

An anomalous value of a numeric attribute (such as *duration*) is yet another interesting event. For every numeric attribute  $B$  in the original raw data, we define new boolean attribute  $O_B$ : "*is attribute B anomalous?*". We compute the average  $\mu_B$  and standard deviation  $\sigma_B$  of the past values and define the new attribute to be true if its value is outside the following interval:  $(\mu_B - 3\sigma_B, \mu_B + 3\sigma_B)$ .

The algorithm described in this subsection (referred to as algorithm  $A_1$ ) was implemented and the results are presented in Section 6.

### Alternate algorithms

In the more advanced algorithm ( $A_2$ ) we would like to correlate the performance and request data. We noticed many short peaks in the performance data that last only for one 5-minute interval; values in the previous and the following 5-minute intervals appear normal. It would be interesting to compare the requests during the *anomalous* interval with the requests during the previous or the following interval. The details of the algorithm are as follows: 1) look at the performance data and detect outliers/peaks, and 2) after detecting an outlier at time interval  $T$ , build a data set from requests during interval  $T - 1$  (class *normal*) and requests during interval  $T$  (class *outliers*). The decision tree algorithm will then try to classify the requests into these two classes; the resulting tree would indicate the most important differences between the classes.

Other extensions include using data from all machines to detect types of requests of particular machines that cause problems and using a better outlier detection algorithm such as a one-class SVM (algorithm  $A_3$ ).

We tried to perform the required preprocessing ad-hoc using Python scripts, but we found that it is very tedious and time consuming to do, as described in the next section.

## 3. Practical problems of log data processing

When presented with terabytes of data, before trying out any SLT algorithm, we need to think about how to handle it efficiently. In this section, we describe our experience with analyzing huge amounts of data. We first describe our early attempts which use ad-hoc Python scripts and traditional relational database systems for handling the data. We discuss the need for a flexible architecture both for processing raw log data and for providing input for SLT algorithms.

### 3.1. Preprocessing data for SLT algorithms

In practice, system log data are in arbitrary formats, and thus, far from ready to be fed into an SLT algorithm. The required preprocessing includes the following:

- **Sampling:** we need to sample the original log because we are not able – and it is not necessary – to look at all the observed data. Other reasons for sampling include temporally variable data rate, dealing with unbalanced data sets, removing duplicate entries or removing entries that do not report the class attribute. Thus, a re-configurable sampling algorithm must be supported.
- **Cleaning the data:** we need to filter out some unnecessary attributes from each of the event log entries: attributes not reported by any sample and attributes with constant values.
- **Adding new attributes:** the original attributes in the raw data are often not enough or not suitable for analysis using SLT algorithms, as described in Section 2.3. Values of some new attributes are easily generated; "*is attribute A reported?*". However, other attributes – such as "*is attribute B anomalous?*" – might require running a simple algorithm.
- **Integrating streams from multiple sources:** System logs are generated on separate machines describing different aspect of the system. For example, our data set contains performance statistics, request log and problem tickets, as described in Section 2.1. It is also often necessary to integrate data sources unanticipated at design time, since we might find more related information as our familiarity of the system increases.
- **Running multiple algorithms:** Applying several different algorithms to our data is an important part of our research. However, different types of algorithms (such as on-line and off-line) require different experiment setup and many publicly available implementations require different format of input data. Because of huge amounts of raw log data, accessing it is a very

expensive operation. Thus, we need to produce a separate output file for each algorithm with one scan of the raw data.

We needed a flexible system that can be configured to generate input data for SLT algorithms in a more convenient way.

### 3.2. Early experience with Python scripts

At the early stages of this project, we did not realize all of the practical problems discussed in the previous section. In order to get started testing the algorithms as soon as possible, we began writing Python scripts to process the data.

Python is a scripting language, which is very efficient (in terms of code length) in processing text files. The simplest preprocessing (scan through the data, project certain attributes of each entry and output them as a text file) can be expressed in about 50 lines of code, and some of our original results were obtained with the data preprocessed in this manner.

However, as we were trying out more algorithms, we found ourselves frustrated by the following problems:

1. **It takes a huge amount of time to scan through the data.** In our case, a single scan through one minute of log data takes more than 10-12 minutes. Most of the time is spent on reading the data from disk, uncompressing it, and parsing it.
2. **It is hard to handle multiple queries in a single script.** Producing data for multiple algorithms in a single script makes the Python scripts significantly more complex. A couple of aggregation queries take about 150 lines of Python code. It became even more complex when we wanted to save some of the intermediate results to disk for future use.
3. **It is hard to add/modify existing queries.** Adding one query to the code may require changing the code for existing queries, because we share the buffer and intermediate results among the queries.
4. **Fine grained parallelism is much harder to achieve.** We observed that preprocessing is CPU bounded (instead of I/O bounded) on our cluster, so speedup can only be obtained by using multiple processors. Parallel processing of the data is very hard to implement in fine granularities (such as at single-request level).

Our experience shows that even though ad-hoc scripts are enough for small data sets, they can be very difficult to maintain if re-running the script is slow and the set of SLT algorithms we need to preprocess the data for is not known in advance.

### 3.3. Problems with relational databases

We also considered using traditional relational databases for our data, since the preprocessing can be specified as SQL queries, reducing the complexity of preprocessing scripts. We rejected this approach for the following reasons:

1. **System logs do not have a fixed schema.** System logs usually have no fixed schema. Efficient relational database operations require a well designed schema; both logical (tables) and physical (file organization, indexing etc.). Changing schema is assumed rare and expensive. However, the format of system logs changes as the system evolves.
2. **One-time-queries are not suitable for generating multiple data output.** The queries in a relational database are *one-time queries*; generating another result usually requires a separate query. If a scan is required on a terabyte data base, each of the queries will take days to run.
3. **It is hard to support queries involving temporal properties of data.** However, this is essential in temporal data analysis.
4. **The cost of using a relational database is high.** Importing multi-terabytes of data into a relational database would have brought to us very high initial cost (both I/O and CPU) and storage cost.

## 4. A flexible architecture

In this section, we introduce a better data model – *data streams* and Telegraph Continuous Query processor to solve the problem described in previous section.

We focus on the functionality and flexibility of our software architecture and experience we get from running a prototype implementation. For the interested readers, we describe the details of the design and implementation of the system in Section 7.

### 4.1. Stream model of system log data

As our early experience suggests, a good data model is necessary for processing huge amounts of system log data. A *data model* is a collection of high-level data description constructs that hides the underlying low-level storage details [16]. It helps people to understand the data better and they can build proper data processing systems to manipulate the data.

We found that system logs fit the stream data models well, because of the following characteristics [2]:

1. Log data entries arrive on-line. The rate is determined by the data source and the temporal rate variation can be large. Also, the stream processor does not have any control over the order in which data elements arrive. In large scale distributed computer systems, such as the system X, logs are generated continuously on each of the machines with different data rate.
2. A data stream is an infinitely long sequence of data elements, but the memory on the stream processor is limited. Once a data element is processed, it has to be either discarded or archived, which makes it hard to locate the element and process it again. This situation fits our application well, since old system events are much less valuable, for failure detection, and even for long term client behavior analysis. Once the desired statistics have been obtained from the raw data, it becomes less important, and people usually do not have a chance to look into the old data again. Typical companies archive raw system log data for a few weeks before discarding them, but the statistics and data representing interesting system events are preserved. Stream processing and SLT can help us find and preserve the interesting data more efficiently.
3. A time stamp is attached to every entry in the data stream explicitly or implicitly (i.e., the arrival time at the stream processing system). Therefore, the system preserves the temporal properties of the data. It also allows most recent data to be accessed more efficiently so that temporal algorithms can be used easily.

The benefits of using data stream over ad-hoc scripts and relational databases are the following:

### Continuous queries

The queries on data streams are usually *continuous queries*, in contrast to one-time-queries. One-time-queries run on a snapshot of the data, and return a single result to the user, while continuous queries are evaluated as data elements in the stream arrive.

Here is an example of a continuous query: *Which machine has handled 5 times more requests than any other machine over the last 10 minutes?* This query has to be re-evaluated each time a new system log entry arrives, and thus produces an output data stream containing the name of machines. The output stream can be used directly as an indicator of system failures or can be used as a regular data stream, for example, as an input to an SLT algorithm. Continuous queries can either be pre-defined or ad-hoc, and multiple queries can run on a single data stream concurrently. The ability to perform continuous queries has great advantage for preparing data for SLT algorithms.

### Off-line SLT algorithms can be used too

As described in Section 8, many commonly used SLT algorithms are off-line algorithms (e.g., our decision tree algorithm), which work on *chunks* of data instead of streams. It is trivial to accumulate preprocessed stream in a buffer to get a data chunk large enough for the off-line algorithm. It is better than traditional methods in that preprocessing the data *before* buffering it allows us to save only the data we want, thus making the buffering much more efficient.

### Easy-to-change schemas

It is easy to change stream schema, since there is no data stored in database. This makes it easy to add new streams and modify the output desired. For example, to generate the data for algorithm  $A_2$  (Section 2.3) we needed to integrate performance and request data.

The idea of stream processing has been commonly used in the past. For example, *grep* in UNIX is a program that processes stream queries specified as regular expressions. The Python scripts we wrote are also stream processors. However, they resulted in ad-hoc and complex solutions.

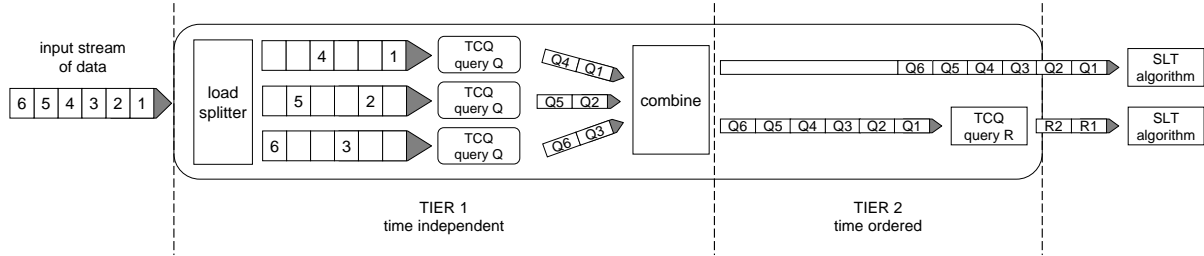
## 4.2. Overview of the architecture

We wanted to build an infrastructure to support data analysis research of system log data. A major concern is *simplicity*. It should be simple enough that the initial configuration should either be automatically generated or be specified with a high-level description. The interface between our architecture and the system monitored should allow *easy deployment* in production environment. This architecture should be *flexible* enough to accommodate many algorithms, both on-line and off-line, without significant re-configuration. It should also be easy to add or remove data streams and components.

The purpose of the system is also to *make the algorithm implementation as easy as possible*, so that SLT researchers can focus on the algorithm rather than on tedious job of accommodating various input formats of raw data.

We tried to *make use of available software* from other research projects. The major component we use is Telegraph Continuous Query engine (TCQ) [12, 18]. TCQ is a continuous dataflow processing engine build on the code base of PostgreSQL, a popular open-source object-relational database system. It contains functionality of both relational database and stream processing. It supports continuous queries over streams, the cost of query evaluation is shared among all queries and the executor adapts to the characteristic change of the streams. It overcomes the problems with relational databases discussed in Section 3.3.

The use of TCQ helped us to easily specify and add/remove continuous queries, which solved the second and third problem discussed in Section 3.2.



**Figure 1.** A general structure of the system. We used publicly available TCQ implementation as our major building block, and other components are currently written in Python. Our data are originally stored in a single data file. To make the data processing rate as fast as possible, we used 4 machines as stream sources, and load splitter is used to split the the stream round-robin by time to a set of (up to) 36 identical TCQ instances. Before splitting, a unique sequence number is assigned to all the entries of the log to allow a later reconstruction of the original order. The first tier of TCQ nodes performs queries that are independent of time (i.e. queries that do not have a time window specification). The output streams are directed to the stream combiner to reconstruct the time order. If many output streams are required, multiple combiners can be implemented. After the streams are combined, the second tier of TCQ instances performs time dependent queries. The final output streams are output to SLT algorithms. The output from SLT algorithms is also modelled as a data stream which can be displayed in a GUI and/or redirected to a centralized controller as feedback from the system.

*Turn-around time* is our next concern, or more specifically, the delay before one can start evaluating an SLT algorithm. We organized our architecture in multiple tiers and run each tier in parallel. Our software architecture is build on TCQ, which has all benefits of TCQ and it allows user to specify fine-grained parallel execution over a computer cluster to achieve short turn-around time and scalability. The main features include:

1. Data in the system are modelled as data streams, which are easy to understand and manipulate. Design of the system is driven by the flow of data. The output of one stream processor can be used as input of another and any stream can be buffered and used by an off-line algorithm. Another advantage of using streams, is that users unfamiliar with SQL can simply specify their queries in other languages such as Java.
2. It is easy to buffer a stream of data for a certain period of time to support off-line algorithms that require chunks of data. Result-saving policy can be specified separately for each stream in order to deal with temporal variation of stream data rate, importance of different streams and storage constraints.
3. It is also simple to cache/store any intermediate stream to disk and reuse it later. This is especially important for research purposes, as we are constrained by the hardware resources available to us.

The architecture also supports the functionality described in Section 3.1, which we present with an example in the following section.

## 5. Experience with our prototype

We implemented a prototype of software architecture on a local cluster. The prototype is designed to be modular enough so each component can be easily replaced as long as it follows a simple stream interface. Major components include: data source, load splitter, stream combiner, TCQ processor, and SLT algorithm wrapper. The structure of the system is easily specified in an object oriented way (description of the components and the interconnections among them), and then automatically translated into a sequence of scripts which start all the components on multiple nodes of the cluster.

The general structure of the architecture is described in Figure 1. The functionality and interface of these components are described in more detail in Section 7. Here we focus on how to use the system and the benefits of using it, which we believe is of more interest to our readers.

This example uses two tiers; the first tier performs time-independent sampling and processing that reduces the size of the stream for time-dependent processing. The data are currently processed in the order they enter the system for the ease of implementation. We consider more advanced load balancing as an important part of our future work.

### A simple query

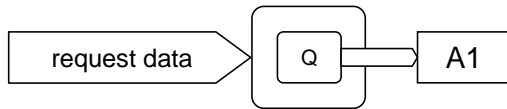
We start with a simple query from a traditional monitoring system; query R: *the average latency for HTTP requests over last 30 seconds*. Note that this query cannot be run on TCQ instances of tier 1 (query Q on Figure 1), since not all the data for "last 30 seconds" is present in either one of the queries Q. Instead, query Q just performs sampling to decrease the size of the stream (time independent) and the

```

select avg(f_delay) as f_delay_avf,
       stddev(f_delay) as f_delay_stddev
from rawlog
where f_app='http'
group by f_app window r ['30sec'];

```

**Figure 2.** A continuous query that computes the average latency for HTTP requests in the input stream over the last 30 seconds. This query can only run on tier 2 in Figure 1, since it involves time-dependent queries.



**Figure 3.** System architecture for algorithm  $A_1$ . The outer rounded rectangle corresponds to the whole architecture from Figure 1. For this algorithm the input stream consists of 450 attributes. If an attribute is not present in the entry, its value is defined as NULL.

actual query  $R$  is executed *after* the combiner in the second tier of TCQ instances. Adding a query (as specified in Figure 2) only takes one SQL statement and running a single-line shell command and can be done on-line without stopping the stream sources.

### Supporting Algorithm $A_1$

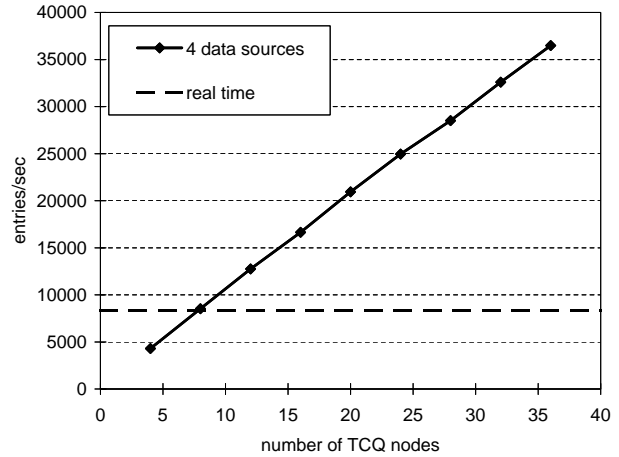
Figure 3 presents the simplified system configuration for algorithm  $A_1$  (Section 2.3). In particular, we want to generate a decision tree for attribute *f\_class* for requests from machine *machine54* and application *proxy*. Query  $Q$  on Figure 4 shows all the required preprocessing. The output stream generated by  $Q$  is formatted as comma-separated-values and can be directly used in a decision tree implementation. Note that this stream must be buffered to form a chunk before passed to the algorithm. The description of the system takes 48 lines of code.

```

select f_class, f_app, f_machine, f_bytes-served, f_ip,
       (f_error-code is not null) as f_error-code-reported,
       (f_duration < LOW OR HIGH < f_duration)
       as f_duration_outlier
from rawlog
where f_app='proxy'
      f_machine='machine54'
      f_class is not NULL

```

**Figure 4.** Query  $Q$  for algorithm  $A_1$ : the first line projects the specified attributes and the next two lines generate two new attributes (Section 2.3) from raw log. The where clause specifies that only accepted entries are from machine *machine54* and application *proxy* that also report a value for the class attribute. Note that we prefer to let this query run on tier 1 (Figure 1), since the output stream is much smaller than the input.



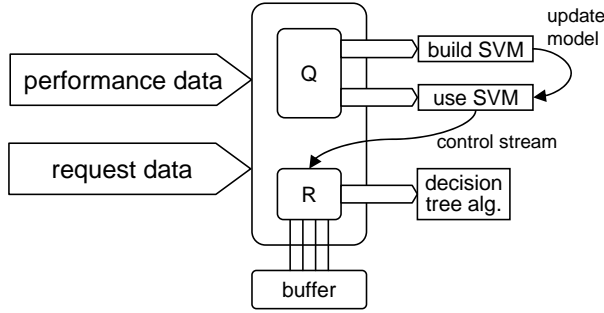
**Figure 5.** When increasing the number of TCQ instances running in parallel, average throughput increases linearly. System  $X$  generates about 8360 log data entries per second which can be handled on 8 TCQ nodes in real-time. 36 TCQ nodes can handle 36500 entries per second which is  $\approx$  600GB of log data per day. We used 4 parallel data sources; with fewer data sources, their throughput will eventually become bottleneck of the system.

The speedup when using multiple machines in parallel is shown in Figure 5. We used 4 parallel stream sources (i.e., we split the log file containing data for all of the 430 machines into four pieces and used 4 nodes to generate stream). Running up to 36 TCQ instances in parallel, we observed linear speedup and the system processed 1 minute of the system log (501572 log entries) in 13.7 seconds. We think that this turn-around time is good enough for us. Also, since the data are produced in time order, we can run algorithms on the output streams without waiting for the processing to complete. In a real world implementation, the data source itself is naturally parallel since the logs are generated on different machines.

### Supporting algorithms $A_2$ and $A_3$

To support algorithm  $A_2$ , we need to add the performance data stream to the system. This can be done by simply adding a data source and since the data rate is very low, a load splitter is not required. We connect this stream directly to one of the TCQ instances in the second tier, after the first tier processing is done. This TCQ instance looks at the performance metrics and uses their average and standard deviation to detect outliers. The output stream of this query (called a *control stream*) is non-zero when an outlier is detected. The control stream is then connected to another TCQ instance in the first tier that generates an appropriate stream for  $A_2$ .

The architecture for  $A_3$  presented on Figure 6 is very



**Figure 6.** structure of the architecture that supports  $A_3$ . The data stream is used to build a Support Vector Machine (SVM), which is an off-line algorithm. the SVM built is then used on-line to detect outliers. Once an outlier is detected, a new query is initiated so that the data from last 10 minutes are preprocessed for decision tree algorithm C4.5 as described in Section 2.3.

similar to  $A_2$ . The outlier detection is now performed by an external algorithm – an off-line one-class SVM. The performance data stream is thus forwarded to two components: one that generates the model and another one that use a previously generated model for outlier detection. The rest of the architecture remains unchanged. These simple changes to the architecture for  $A_1$  show the flexibility of the system.

### Supporting any algorithm

Since the data flows in streams that can be buffered we can use this architecture with any on-line or off-line data mining or SLT algorithm. All entries entering the system contain a time stamp and, in addition, we assign them a unique sequence number. All the processing (such as SQL queries or merging different data streams) preserves the original order of entries and thus temporal algorithms can be used too. Further, because the output of any algorithm is also modelled as a stream, we can arbitrarily combine multiple algorithms.

### Updating a component of the system

In our initial setup, we used a very simple stream source component which was implemented using a single-threaded Python script. After the system was up and running, we found its performance unsatisfactory, so we implemented a new stream source with multi-threading. Removing the old source and adding the new one took only two shell commands (one to stop the old one and the other to start the new one), while all other components were left intact without restarting. The whole process took only a couple of minutes, before the system continued to produce new output data. All the stream source implementations are straightforward and even the multi-threaded one consists of only 90 lines of Python code.

## 6. Results of algorithm $A_1$

In this section we present results of the algorithm  $A_1$  from Section 2.3. We used the standard implementation of C4.5 decision tree algorithm from the machine learning tools package Weka [23]. The depth of the trees was constrained to be no more than 7 and the minimal number of samples in a leaf was set to 10.

The results are based on 4 hours of request data from a single machine running mainly the proxy application. The following preprocessing was applied: a) add new attributes as described in Section 2.3, b) remove useless samples (ones that did not report the class attribute), c) remove useless attributes (ones that are constant), and d) resample the data to obtain smaller data set (about 20 - 30 MB) and to balance the classes of the class attribute. The resulting trees are presented in Figure 7. The decision trees  $B'$  and  $C'$  were generated from the original data set after filtering out attributes `R.cache-served` and `duration`, respectively.

### How useful this is for the operators

As this is work in progress, we didn't yet carry out any extensive evaluation of this algorithm. However, the early results are encouraging; we showed these decision trees to one of the operators in company X and he thinks they are very useful.

For the operators, there are two main characteristics that make this approach very valuable. First, the decision tree algorithm can automatically search through the large number of attributes and find a small subset that is correlated with the class attribute. For example, the first decision tree for attribute `R.error-code` says the following: *"the requests that report an error-code (almost certainly) do not report attributes cache-served and server-duration"*. The structure of the decision tree also allows the operator to quickly identify points of interest: for example URLs `6520...` and `2336...` (see the decision tree  $B'$ ) almost always generate an `error-code`. The possibility of defining new attributes on-the-fly makes this very attractive.

We can hardly expect an automated analysis system to replace the operators; an ideal system will thus accept feedback from the operators and improve its analysis accordingly. Decision trees allow a simple version of this: if the decision tree generates rules that are trivial for the operator and he understands them, a few attributes can be filtered out from the data set and the algorithm can generate an alternative explanation.

An example can be seen in the decision tree for attribute `O.client-write-duration`; the first decision tree generated a simple decision tree that is probably clear for the operator (*"attribute duration is almost perfectly correlated with the outliers in attribute client-write-duration"*). After removing the attribute `duration` from the data set, the alternative decision tree offers more interesting insights.



blocks (see Figure 1) that communicate with each other using sockets. They can be deployed on a single physical node or over multiple nodes in a cluster. These components were written in Python and comprise about 750 lines of code.

**Data source** is the interface for getting the various kinds of data, translating them into data streams and feed them into the stream processing system. It provides a small interface to the production system, and can be overridden to use multiple types of data, such as logs stored on disk, network monitoring readings, or live stream of system event reports.

**Load splitter** is a small component used for load balancing that takes a single input stream, divides it into multiple streams and redirects them on to multiple nodes. When the data rate cannot be handled by a single TCQ instance, we create multiple instances and use the load splitter to route the stream to all the instances. The data processing within the load splitter should be simple and fast, since it is on the critical path of the system and always sees a large data rate. Currently we think the best algorithm is splitting by time, i.e. sending data elements in round-robin manner to each of the stream processors. This provides best throughput since there is no cost for examining and parsing the data. Of course, more advanced load splitter can be built, especially considering the properties of the source stream. Load splitter can be changed in the system without affecting any other components.

**Stream combiner** is a component used to combine multiple streams generated by load splitter to re-create the original order of entries. It works on the original time-stamp attached to each entry in the stream. If the time-stamp is too coarse grained to order the entries (for example, there are 600 events in a single second and some of them have causal dependency), we attach a unique sequence number to each entry when it is pushed into the system.

**TCQ instances**, as described in Section 7.1, are the key components in the system. They take in multiple SQL queries, multiple data streams and output the results of the queries as data streams. The output data streams can be buffered for off-line algorithms or written to files for future use. The raw stream can be configured to be archived. The TCQ instances also output its own performance statistics as data streams (which is called introspective query) to centralized controllers.

**SLT algorithms** are defined as a components taking data streams or a file as input and output another data stream. The data stream can be interpreted by a GUI component for human administrators to review or achieved for future reference. Publicly available algorithms can be plugged into the system with only a minor wrapper for reading data streams (for example, through JDBC or simply through a UNIX pipe).

Note that the data streams between each component are not necessary in the same format. They can be implemented

as text streams, but we can also use binary streams with type definition which saves parse time. Changing the format of a stream is simple; it only requires to change the output format of the sender and input format of the receiver or add a separate wrapper around the receiver.

### 7.3. Putting everything together

We provided a simple way to build the system from components with simple object oriented specifications. All the components are modeled as a class. To build the system, one only needs to specify the parameters of the components or override some of the functionalities and the interconnections among them. A program we provide automatically generates a shell script that starts the components on multiple machines in the cluster. There are separate scripts for adding and removing streams and queries from each.

There are two steps to add a new SLT algorithm. First, the algorithm "subscribes" a stream from the system, which is done by specifying a new query. Second, the user may need to write a simple wrapper to generate the correct format and/or add a header.

We feel that although now it takes only less than half an hour to write the script that generate the architecture in Section 5, it would be more convenient to use a GUI.

## 8. Related work

This research is a multi-disciplinary research involving SLT, data mining, dependable system design, data warehousing and processing.

### SLT for systems

Recently, a few researchers started using SLT algorithms for detecting and localizing system failures and software bugs.

In [4], Chen uses decision trees for localization of failures on the eBay web site. Each executed request reports attributes such as name, type, machine, version and status of the request. A decision tree is trained to predict the status attribute and the generated rules are used to localize what machine, type of request, or version of software is causing problems.

In his work on Pinpoint [3, 9], Kiciman instrumented the JBoss application server so that a J2EE application reports *execution paths* of all requests. The path is a list of J2EE components that the particular request used. Pinpoint can detect anomalous paths and correlate them to identify the failed components.

Cohen *et al.* [5] uses Tree-Augmented Bayes Nets for automated performance analysis. 124 types of performance metrics are measured on a sample server and the induced model is used for prediction of Service Level Objective violation.

Researchers at IBM Research [21, 22, 17] apply temporal data mining and time series analysis to predict critical events in computer system such as *high CPU utilization* or *imminent router failure*.

Liblit [10] proposes a sampling infrastructure for gathering information about execution of C programs. He instruments the source code of the program at every branch, assignment and function call. The recorded information from runs of the program that crash are correlated to obtain the possible bugs.

Most of the work mentioned above has the same goal as our research – use automated analysis of computer systems. However, experiments conducted in the referenced papers use smaller data sets (10 – 100MB) compared to our data set of a few terabytes.

### **System monitoring and management**

There have been a lot of efforts on monitoring systems in both academia and industry. Simple Network Management Protocol (SNMP) [8] allows user to instrument and monitor aggregated performance of heterogeneous component in a network environment. It provides a visualized and hierarchical infrastructure to support high volume data collection and separating management boundaries.

There are commercial tools that allow user to monitor and do simple analysis on the data collected. The major tools include HP OpenView (<http://www.openview.hp.com/>), IBM Tivoli (<http://www-306.ibm.com/software/tivoli/>), Microsoft Operations Manager (<http://www.microsoft.com/mom/>). These tools allow user to navigate through the collected and stored data, and run statistical analysis on them. However, they are not designed for preparing data for SLT algorithms.

Traditionally, the collected data are sent to some centralized servers which may waste bandwidth. Both Astrolabe [20] and PIER [7] manage to collect and analyze the data on the node where they are generated. Astrolabe makes use of gossip protocol and the architecture is formed in a hierarchical structure of domains. PIER is implemented on a DHT [19]. Both allow user to run queries in SQL which are then evaluated in a distributed way in the system.

### **Stream data processing and mining**

Our work is also related to the stream processing and data mining work in database community. Stream processing addressed the issue of dealing with data that arrive in multiple, continuous, rapid and time-varying data streams [2].

A number of stream processing systems have been proposed to handle continuous queries over a data stream [2, 1]. TelegraphCQ [12] addresses this problem with eddy query processing framework that adapts the temporal variation of data streams in data rate and statistical characteristic of the data stream. It also allows to share evaluation path among multiple queries.

Several new algorithms that are suitable for mining data streams were proposed. The characteristic of most of these algorithms is that they only look at every tuple in the stream once [6]. In contrast, for most of the SLT algorithms it is not enough to look at each data tuple just once. Buffering and caching of old data are supported in our work to solve this problem.

Stream processing is also used in sensor network data monitoring and analysis [11]. Though the data rate from sensor network can also be high, it is much less complex than logs generated by a large cluster of computers.

## **9. Future work**

As a joint disciplinary research project, we have two tracks of future research that are interleaved.

### **SLT algorithms**

There are a few interesting challenges for SLT researchers. To understand a system better, we can – in the extreme case – instrument every line of source code of the applications running on the servers. However, this instrumentation is highly redundant and so an interesting question to ask is *“how much instrumentation do we need to get the best results?”* Source code instrumentation at even a much coarser level will generate a few orders of magnitude more attributes; *“how can we select a subset that contains the most interesting attributes?”*

Another problem of current SLT algorithms is that they assume global knowledge of all the data. However, with tens of thousands of machines, the central storage will certainly become a bottleneck; we cannot even download all the data. The possible solutions include: a) store history of the raw log data in the machines, monitor only a few selected attributes and download the necessary data only after we detect a problem, b) do early processing of the log data in the machines, or c) sample all the attributes and over time decide which ones are less/more important and sample slower/faster.

### **Software architecture**

We will turn the prototype discussed in the paper into a more general toolkit and make it publicly available to the SLT researchers. There are several other components that we plan to add into our system, which are not implemented in the prototype.

- We want to support the operator-automated system interaction. We want to implement this by a GUI for viewing the output of the algorithm, modify the parameters and data fed into the algorithm to generate better result. We also want to model human input as another stream into the system and let the system adjust queries automatically.

- Adding a centralized controller to the architecture, which collects statistics from different components and dynamically allocates physical nodes to each of the components. This gives us the ability to monitor and analyze the architecture itself. We can monitor the load of each component of the system and dynamically balance load when sudden load change happens or query behavior changes.
- A distributed optimizer for query evaluation. Currently adaptive query executor only works on single node, but we need a centralized/distributed query optimizer in order to remove the job of routing the stream manually among the nodes, and thus providing a single system image of the distributed architecture.

## 10. Conclusion

Our experience suggests that we cannot use ad-hoc systems for automated analysis of huge system log data set from on-line services. Instead, we propose a modular architecture that was shown to easily handle 600 GB of system log data a day. The architecture is flexible enough to be used for any type of on-line or off-line data analysis algorithm.

## References

- [1] C. C. Aggarwal. A framework for diagnosing changes in evolving data streams. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 575–586. ACM Press, 2003.
- [2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM Press, 2002.
- [3] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic internet services. DSN 2002.
- [4] M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer. A statistical learning approach to failure diagnosis. In *International Conference on Autonomic Computing (ICAC-04)*, New York, NY, May 2004.
- [5] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. Chase. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *OSDI*, 2004. To be published.
- [6] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: you only get one look, a tutorial. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 635–635. ACM Press, 2002.
- [7] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the internet with PIER. In *Proceedings of the 29th VLDB Conference*, 2003.
- [8] M. S. J. Case, M. Fedor and J. Davin. A simple network management protocol (SNMP). *RFC1157*, May 1990.
- [9] E. Kiciman and A. Fox. Detecting and localizing anomalous behavior to discover failures in component-based internet services. Technical report, Stanford, 2004.
- [10] B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan. Bug isolation via remote program sampling. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, San Diego, California, June 9–11 2003.
- [11] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM Press, 2003.
- [12] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 49–60. ACM Press, 2002.
- [13] J. Markoff and G. P. Zachary. In searching the web, Google finds riches. NY Times, April 13, 2003.
- [14] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do internet services fail, and what can be done about it? In *4th USENIX Symposium on Internet Technologies and Systems*, March 2003.
- [15] D. A. Patterson. A simple way to estimate the cost of downtime. Submission to 16th Systems Administration Conference (LISA '02), 2002.
- [16] J. G. Raghuram. *Database Management Systems*. McGraw-Hill Higher Education, 2003.
- [17] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, and S. Ma. Critical event prediction for proactive management in large-scale computer clusters. In *KDD*, 2003.
- [18] S. K. Sirish. Telegraphcq: An architectural status report.
- [19] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In R. Guerin, editor, *Proceedings of SIGCOMM-01*, volume 31, 4 of *Computer Communication Review*, pages 149–160, New York, Aug. 27–31 2001. ACM Press.
- [20] R. van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, 2003.
- [21] R. Vilalta, C. V. Apte, J. L. Hellerstein, S. Ma, and S. M. Weiss. Predictive algorithms in the management of computer systems. *IBM Systems Journal*, 2002.
- [22] R. Vilalta and S. Ma. Predicting rare events in temporal domains using associative classification rules. Technical report, IBM Research, T. J. Watson Research Center, Yorktown Heights, NY, 2002.
- [23] I. H. Witten and E. Frank. *Data mining: practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann Publishers Inc., 2000.

# Discovering Target Events Rules based on Time-Consecutive Pattern Mining\*

Ehud Gudes  
ehud@cs.bgu.ac.il  
Ben-Gurion University of the Negev  
Beer-Sheva, 84105, Israel

Litvak Marina  
litvakm@cs.bgu.ac.il  
Ben-Gurion University of the Negev  
Beer-Sheva, 84105, Israel

## Abstract

*We are given temporal customer-oriented dataset, where each transaction consists of set of events that are associated with a customer id and a timestamp. For each customer there are several transactions with different timestamps. One of the events is defined as the target event. We introduce the problem of mining target events rules that are based on the discovery of continuous sequential patterns over such databases. We propose two algorithms to solve this problem and evaluate their performance using real-life data. The presented algorithms, CTSPD and CSPADE, have comparable evaluations, but the CTSPD, as expected, turns out to work faster.*

## 1 Introduction

**Motivation** Many of the the modern databases are temporal and customer-oriented, like the famous basket database, that is, a dataset of purchases where each purchase is associated with an owner and a time [3, 8, 2]. Such databases collect and store the transactions in the order of receiving the data. A huge amount of data is collected every day in the form of event time sequences. Common examples are recordings of different values of stock shares during a day, accesses to a computer via an external network, bank transactions, or events related to malfunctions in an industrial plant. These sequences register events with corresponding values of certain processes, and are valuable sources of information not only to search for a particular value or event at a specific time, but also to analyze the frequency of certain events, or sets of events related by particular temporal relationships. This type of analysis can be very useful for predicting the future behavior of the monitored process.

Many commercial problems for data mining include prediction of the behavior of the customers. The miner can define the target predicted event, like the customer will/will

not buy something or will/will not change his/her status. The companies are usually interested in the sequence of actions or events that leads to such particular action (let's note them *basic* events and *target* event, respectively) [9]. For that purpose such companies may collect and store the information about customers in the form of transactions including customer id, time, set of basic events and the target event associated with that id and time. We introduce the problem of target events rules using continuous sequential pattern mining over such dataset.

A sequential pattern is defined as  $A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_n$  where  $A_i$ ,  $1 \leq i \leq n$  are events (or set of events) and  $\text{time}(A_i) \leq \text{time}(A_j)$  for  $i \leq j$  [6, 15, 5, 1].

In the previous algorithms for mining sequential patterns the resulting sequence need not be continuous (the sequence  $AB \rightarrow C$  is continuous if there is no event  $D$  that happens after  $AB$  and before  $C$ ). But such patterns do not contain information about how exactly the pattern's events affect its other events. For instance, the pattern  $A \rightarrow B$  may express " $B$  happens year after  $A$ , and some other events happened during that period" and, at the same time, it can mean something different, like " $B$  happened immediately after  $A$ ". We are interested only in precise, continuous sequences that imply a target event. An example of such a rule may be: "After buying a ticket to the football match which was immediately after purchasing a merchandise, the customer buys a sport lottery". We will give a precise definition of these rules later in this paper.

The application that we are interested in and have real data on is that of up-sale/cross-sale of telecommunication companies that are interested in selling additional or up-graded products to their customers [9]. Companies need to identify target-customers for product upgrade sales. One important potential source of information is usage data, e.g. how much bandwidth was used by each user at every hour during the last weeks or months. Customer usage data differs in that it is a time series. Although there is a considerable amount of work on time series, most of it deals with the prediction of a future value in the time series, as in stock market prediction [11]. In our case, we use the time se-

\*partially supported by the Lynn and Frankel center for computer science and by the Paul Ivanir Robotics center.

ries in order to predict a phenomenon external to the time series. Our target concept is a boolean feature that indicates whether the customer will buy an upgraded product in the near future. To reduce the problem to the attribute-based, we extract the subset of important threshold features of the form: the fraction of the time in the period  $X$  that a certain measure was above  $Y$  (for example, such feature may be the fraction of the time that the bandwidth utilization was above 90% in the last 5 weeks.) Since all the features were continuous, we used abstraction algorithms to map them to discrete values. The reduction of the problem into attribute based allows us to use conventional data mining algorithms (with our enhancements). In [9] were used rules received from such algorithms as Association Rules (including Quantitative and Clustered Association Rules) and BKB.

**Related work** The problem of sequential patterns was introduced in [5] by R. Agrawal and R. Srikant. They presented three algorithms for solving the problem of discovering sequential patterns over large databases of customer transactions. The proposed algorithms generate a data sequence for each customer from the database and search on this set of sequences for a frequent sequential pattern. For example, the algorithms can discover that customers typically rent “Star Wars”, then “Empire Strikes Back”, and then “Return of the Jedi”. Two of these algorithms, GSP and AprioriSome, were designed to find only maximal sequential patterns and the third algorithm, AprioriAll, finds all patterns. Briefly, AprioriAll is a three-phase algorithm. It first finds all itemsets with minimum support (frequent itemsets), transforms the database so that each transaction is replaced by the set of all frequent itemsets contained in the transaction, and then finds sequential patterns. All algorithms are based on the classic Apriori principle, introduced with the Association Rules Discovery problem [3], [4].

In a later paper, the same authors, [13], generalize the problem by introducing such terms as time constraints that specify a minimum and/or maximum time period between adjacent elements in a pattern, and a user-defined taxonomy (is-a hierarchy) on items allowing sequential patterns to include items across all levels of the taxonomy. A sequence in [13] consists of a list of sets of items, rather than being simply a list of items. In addition, authors of [13] are interested in finding all sequences with minimum support rather than only maximal frequent patterns.

H. Mannila et. al. [12] search event sequences for frequent patterns of events. These patterns have a simple structure (essentially a partial order) whose total span of time is constrained by a window given by the user. The technique of generating candidate patterns from sub-patterns, together with a sliding window method, results with effective algorithms. Similarly to [5], the strategy of [12] is starting with

simple sub-patterns (subsequences in this case) and incrementally building longer sequence candidates for the discovery process.

The work by Wang et. al. in [14] also deals with the discovery of sequential patterns, where the considered patterns are in the form of specific regular expressions with a distance metrics as a dissimilarity measure in comparing two sequences. The proposed approach is mainly tailored to the discovery of patterns in protein databases.

M. Zaki, [15], analyzes sequential dependencies among different events and introduces an algorithm that is called SPADE (Sequential Pattern Discovery using Equivalence classes). For efficient and fast search for all frequent patterns, SPADE utilizes combinatorial properties to decompose the original problem into smaller sub-problems, that can be independently solved in main-memory using efficient lattice search techniques, and using simple join operations. The ‘search space’ is decomposed into prefix-based parent equivalence classes and all frequent sequences are enumerated via BFS or DFS search within each class. All sequences are discovered in only three database scans.

In order to solve the event prediction problem, Gary M. Weiss in [10] developed Timeweaver, a genetic-based machine learning system that, given a pre-specified “target” event, learns to identify patterns in the data that successfully predict the future occurrence of that event.

In our work we use some of the above methods but adapt them to the special application domain where we are interested only with rules whose right side is a target event and left side is a consecutive sequential pattern which ends with the target event.

**Organization of the paper** We give a formal description of the problem of mining target events rules in Section 2. In Section 3, we describe CTSPD, an apriori-based algorithm for finding such rules from continuous patterns, and present an illustrative example. In Section 4 we describe the second algorithm (CSPADE) which is based on SPADE [15], and is modified according to our problem definition. In Section 5 we empirically compare the performance of CTSPD with the CSPADE, and study their scale-up properties. We conclude with a summary in Section 6. The appendix contains a more detailed description of SPADE [15].

## 2 Definitions and Problem Statement

We are given a dataset  $D$  of customer transactions. Each transaction consists of customer id, timestamps (start date and end date), several basic events and one target event. More formally, such transactions look like  $\langle cust\_id, s\_date, e\_date, ev_1, ev_2, \dots, ev_n, target \rangle$ , where  $ev_i$  is the basic event occurring at  $[s\_date, e\_date]$  and the  $target$  is a boolean feature that indicates whether

the customer did or didn't do some action in the end of this period (for example, did he buy an upgraded product or didn't). Formally, each basic event is associated with the period between  $s\_date$  and  $e\_date$ , and each target event is associated with  $e\_date$ . No customer has more than one transaction with the same timestamps.

An event  $ev_i$  is denoted by a pair  $\langle event\_type_i, event\_property_i \rangle$ , where  $event\_type_i$  is an attribute and  $event\_property_i$  is a value of that attribute in the record.

An *itemset* is a non-empty set of events occurred at the same period. Denote itemset  $l$  by  $(ev_1, ev_2, \dots, ev_n)$ , where  $ev_i$  is an event.

A *sequence* is an ordered list of itemsets and denoted by  $l_1 \rightarrow target_1 \rightarrow l_2 \rightarrow target_2 \rightarrow \dots \rightarrow l_n \rightarrow target_n$ , where  $l_i$  is an itemset of basic events and  $target_i$  is an event with the boolean property yes/no. We call  $l_i$  the  $i^{th}$  *basic level* (level of basic events) and  $target_i$  is the  $i^{th}$  *target level*. Note that  $l_1$  and/or  $target_n$  need not exist — the sequence can start from the target level and/or end on the basic level. The sequence has to be continuous. The sign  $\rightarrow$  means “happened immediately after” (or, more precisely, “without other events occurring in between”).

The sequence  $s$  is denoted by  $l_1 \rightarrow target_1 \rightarrow l_2 \rightarrow target_2 \rightarrow \dots \rightarrow l_n \rightarrow target_n$  is *continuous* if for each  $i$  there is no event  $ev : ev \notin l_i, ev \notin l_{i+1}$  and  $ev \neq target_i$  that happens after period of  $l_i$  and before the period of the  $l_{i+1}$ .

All customer transactions of the same id sorted by the timestamp can be viewed as a single sequence. We will call it a *customer-sequence*. Note, that a customer-sequence will always have the first basic level and the last target. A transaction database  $D$  converted to a collection of such customer-sequences is called *sequence database* and is denoted by  $T_D$ .

The *length* of a sequence is the number of levels in the sequence. Denote the sequence received from the sequence  $s$  by removing the last level by *R-prefix* and the sequence received after removing the first level by *R-suffix* of  $s$  (restricted prefix and suffix accordingly). The *size* of a sequence is the number of events in the sequence. Denote the sequence of size  $k$  by *k-sequence*. The *size* of an itemset is the number of events in the itemset. Denote the itemset of size  $k$  by *k-itemset*. For instance, the sequence  $AB \rightarrow U \rightarrow C$  has length equal to 3 and size equal to 4.  $AB$  is a 2-itemset and  $U$  and  $C$  are 1-itemsets (actually, the target levels are always 1-itemsets). And for sequence  $s$   $A \rightarrow U \rightarrow C$  both length and size are equal to 3.  $A \rightarrow U$  is the R-prefix and  $U \rightarrow C$  is the R-suffix of  $s$ .

Given any continuous and frequent  $k$ -sequence  $s$  denoted by  $l_1 \rightarrow ltarget_1 \rightarrow l_2 \rightarrow ltarget_2 \rightarrow \dots \rightarrow l_n \rightarrow ltarget_n$  and there is  $i$  (at least one) such that  $l_i$  is the  $n$ -itemset with  $n > 1$ . We call such sequence *narrowable* sequence. The

sequence formed by substitution of  $l_i$  for  $l'_i$ , where  $l'_i \subset l_i$  and  $|l'_i| = n - 1$  is called a *narrowed-sequence* of  $s$ .

**Lemma 1:**

- a) All narrowed-sequences of a frequent continuous sequence (if exist) must be frequent and continuous too (reverse is not true).
- b) R-suffix and R-prefix of the frequent continuous sequence must be continuous and frequent too.

The sequence  $s' l_1 \rightarrow ltarget_1 \rightarrow l_2 \rightarrow ltarget_2 \rightarrow \dots \rightarrow l_n \rightarrow ltarget_n$  is *contained* in another sequence  $s k_1 \rightarrow ktarget_1 \rightarrow k_2 \rightarrow ktarget_2 \rightarrow \dots \rightarrow k_m \rightarrow ktarget_m$  (denote  $s' \subset s$ ) if and only if  $\exists j : \forall i, 1 \leq i \leq n$   $l_i \subseteq k_{j+i}$  and  $ltarget_i = ktarget_{j+i}$ .

**Lemma 2:** If  $s' \subset s$  and  $size(s') = size(s) - 1$  then  $s'$  is either narrowed-sequence or R-prefix or R-suffix of  $s$ .

Also, some sequence  $s'$  may be contained more than once in another sequence  $s$ . For example,  $AB \rightarrow U \rightarrow C$  is contained in the  $ABC \rightarrow U \rightarrow CD \rightarrow \bar{U}$ . However, the sequences  $AB \rightarrow \bar{U} \rightarrow C$  or  $AD \rightarrow U \rightarrow C$  are not. The sequence  $C \rightarrow U$  is contained in the sequence  $AC \rightarrow U \rightarrow BC \rightarrow U \rightarrow C$  twice.

**Definition 1:**  $\chi_{s,s'}$  is the number of occurrences of sequence  $s'$  in a sequence  $s$ . A customer-sequence  $s$  *matches* the other sequence  $s'$  if  $s' \subset s$ .

**Definition 2:** The *support* of sequence  $s'$ , denoted by  $sup(s')$ , is a fraction  $\frac{\chi_{T_D,s'}}{|D|}$ , where  $\chi_{T_D,s'} = \sum_{s \in T_D} \chi_{s,s'}$  and  $|D|$  is the number of transactions in the transaction database. A sequence with minimum support is called a *frequent sequence*.

**Definition 3:** *Target rule* is a sequence tailed with any target event.

**Definition 4:** The *confidence* for target rule  $r$ , denoted by  $conf(r)$  is defined as the fraction:  $\frac{\chi_{T_D,r}}{\chi_{T_D,\hat{r}}}$ , where  $\hat{r}$  is the R-prefix of  $r$ . A rule with minimum support and minimum confidence is called a *frequent rule*.

Given a transaction dataset  $D$  as defined above, the problem of mining target events rules using continuous sequential mining is to find all frequent target rules.

### 3 The CTSPD algorithm

Our first algorithm CTSPD (Continuous Target Sequential Patterns Discovery) is based on Apriori and has two special properties:

- i. The structure of sequences and
- ii. The consecutiveness property.

These properties affect both candidate generation and pruning as is shown below. The CTSPD consists of several basic

phases, that transform the transaction database to the sequence database, mine all frequent sequences and then select from them appropriate rules. In this section we give a description for each phase and an illustrative example.

**1. Sorting the database** First of all, we sort the seed dataset by a customer id and start/end time as primary and secondary keys accordingly. The purpose of this action is straightforward transformation phase — transforming the dataset into a set of customer-sequences. An example of the dataset after such sorting can be seen in Figure 1.

**2. Frequent 1-sequences generation** To find the support for events we scan the dataset  $D$ , and count the number of occurrences for each event. At the end of the scan we know all frequent events. We store the frequent events in two separate structures: hash table for basic events (for quick access) and a 2-element array for the target events. We calculate the number of basic frequent events  $N$  and map them to a set of contiguous integers. These integers are used as the indexes for alternate bit representation of the sequences, described below. To represent the itemset we use a bit string of length  $N$ , where the bit  $i$  is set to 1 if the event with index  $i$  is present in this itemset. Example for both mappings are depicted in Figure 2.

**3. Transformation phase** The goal of this phase is the transformation of the given database to the collection of user-sequences (one sequence for a user, that includes all his frequent events in order of occurring). We don't include the non-frequent events to the customer-sequences, based on the Apriori principle, confirming that frequent itemsets do not include non-frequent elements. If all transactions for a customer do not contain frequent basic events at all, we do not create the respective customer-sequence. To store the temporal relations on events we need a special structure for transactions. We store each sequence in form of ordered list of  $plevels$ , where each  $plevel_i$  is the pair  $l_i \rightarrow target_i$ , where  $l_i$  is the  $i^{th}$  basic level and  $target_i$  is the  $i^{th}$  target level. We present the itemset of basic level by a bit-string as was described above. For example, the itemset  $ACD$  will look as 101100 given the mapping from the Figure 2. We use single bit for target level, that is set to 1 if the target = *true* and to 0 otherwise.

The set of customer-sequences is denoted by  $T_D$ .

**4. Frequent 2-sequences creation** Denote the term of basic event by  $bev$  and the target event by  $tev$ . We create 2 types of pairs:  $bev \rightarrow tev$  and  $tev \rightarrow bev$  from the events generated in the second phase. We need not create the pairs where both right and left parts are basic or target (like  $bev \rightarrow bev$  or  $tev \rightarrow tev$ ) because they cannot be continuous and

---

**Algorithm 1** Frequent sequence generation

---

```

Result  $\leftarrow \theta$ ;
 $L_k = L_2$ ;
while( $L_k \neq \theta$ ) {
    Result = Result  $\cup L_k$ ;
     $C_k = \text{GenerateCandidates}(L_k, L_2)$ ;
    SupportCompute( $C_k$ );
     $L_{k++} = C_k$ ;
}

```

---

will not participate in the candidate generation phase. The pair of type  $bev \rightarrow tev$  is represented by one level and for the pair  $tev \rightarrow bev$  we create two levels, where the first one has an empty basic level, and the target of the second level is empty.

**5. The frequent sequences generation phase** In order to find all the frequent continuous sequences we perform multiple passes over the transformed database. In each pass  $k$  we start with the initial set of frequent  $k + 1$ -sequences and create the next potentially frequent  $k + 2$ -sequences, called *candidates*. We find the support for these candidates during the scan of  $T_D$ , and at the end of the scan we know which of them are frequent. The set of these new frequent sequences becomes the initial set for the next  $k + 1$  pass. We denote the initial set by  $L_k$  and the new candidates by  $C_k$ . This is shown in Algorithm 1. The procedure *CenerateCandidates* uses two different joins in candidates generation and pruning as described below.

**6. Rules generation** Rules generation is the last phase of the algorithm where we filter the sequences found in phase 5. The first filter passes sequences ended by the target level. Then, for each one we calculate the confidence, according to the formula. We remove all sequences having confidence less than *min\_conf*.

---

**Algorithm 2** GenerateCandidates( $L_k, L_2$ )

---

```

 $C_k \leftarrow \emptyset$ ;
for ( $i = 0$ ;  $i < L_k.size$ ;  $i++$ )
    for ( $j = i + 1$ ;  $j < L_k.size$ ;  $j++$ )
        ExpJoin ( $L_k[i]$ ,  $L_k[j]$ );
for ( $k = 0$ ;  $k < L_2.size$ ;  $k++$ )
    ConcatJoin ( $L_k[i]$ ,  $L_2[k]$ );

```

---

**3.1 Candidate Generation**

To generate the next potentially frequent continuous sequences we use two different joins (see Algorithm 2). The

first one joins the frequent  $k$ -sequences from the previous pass, and is called Expanding Join (ExpJoin). This procedure is responsible for *expansion* of the sequences. By *expansion* we denote the process from which two seed  $k$ -sequences of the same length, produces a new  $k + 1$ -sequence (of the same length). It may happen by extending one of the basic levels. This procedure is given two sequences,  $s_1$  and  $s_2$ , checks if their lengths are equal, and if there is a pair of plevels which is appropriate for the Expanding Join given all other plevels are identical. This expansion is similar to itemset expansion in Apriori (see Algorithm 3 and also 4). For example, two sequences  $AB \rightarrow U \rightarrow C$  and  $AD \rightarrow U \rightarrow C$  form the candidate  $ABD \rightarrow U \rightarrow C$ . This technique doesn't protect us from the repeating generations of the same candidates, and to avoid this before inserting the candidate to  $C_k$  we need to check if it's already contained.

The second join, called Concatenating Join (ConcatJoin), concatenates a  $k$ -sequences from  $L_k$  with 2-sequences from  $L_2$ . As a result from this procedure, we receive the new  $k+1$ -sequence of length  $m+1$ , where  $m$  is the length of the seed  $k$ -sequence. The procedure receives two sequences,  $s_1$  and  $s_2$ , and compares  $s_1$ 's last level to the first one of  $s_2$  and  $s_1$ 's first level to the last one of  $s_2$ . As a result, we can generate a maximum of two new candidates from a single pair. This is shown in Algorithm 5. For example, the sequences  $A \rightarrow U \rightarrow B \rightarrow U$  and  $U \rightarrow A$  are appropriate for double concatenation. The two new candidates are:  $A \rightarrow U \rightarrow B \rightarrow U \rightarrow A$  and  $U \rightarrow A \rightarrow U \rightarrow B \rightarrow U$ .

### 3.2 Pruning

In the Pruning stage we remove all sequences that are not continuous and/or not frequent. As we mentioned in the Lemmas of Section 2, all  $k - 1$ -subsequences of a frequent and continuous sequence must be frequent and continuous. To save time by not counting the support for the non-frequent and/or non-continuous sequences, we prune the candidates which do not support this check. For narrowable sequence we generate narrowed subsequences. For non-narrowable sequences we generate two subsequences: R-prefix and R-suffix. The procedure is described in Algorithm 6.

### 3.3 Support counting

In order to check frequency of candidates, we scan the sequence database  $T_D$  and count the number of sequence occurrences in the user-sequences (procedure SupportCount). The procedure used for support calculating and for checking the candidate continuity, is procedure Match described below. It receives two parameters: user-sequence and candidate, and returns number of occurrences of this

---

#### Algorithm 3 ExpJoin( $s_1, s_2$ )

---

```

lev ← -1;
if (s1.length == s2.length){
  for (i = 0; i < s1.length; i++){
    if (s1.pleveli ≠ s2.pleveli){
      if ExpJoinAppr(s1.pleveli, s2.pleveli){
        if (lev == -1)
          //first and uniquely matching
          lev = i;
        else return;
      }
      //there is more than one matching
    }
    else return;
  }
  //the plevels are not equal
  and not appropriate
}
Candidate = ExpandLevel(s1, s2,
plev);
//plev denotes where the expansion
occurs
if (!Prune(Candidate))
  Ck = Ck ∪ Candidate;

```

---



---

#### Algorithm 4 ExpJoinAppr(plev1, plev2)

---

```

if(first k-1 bits at the
basic levels are the same
&& kth bits are different
&& target1 == target2)
  return true;
else
  return false;

```

---



---

#### Algorithm 5 ConcatJoin( $s_1, s_2$ )

---

```

if(s1.lastLevel == s2.firstLevel){
  Candidate = concat(s1, s2);
  if (!Prune(Candidate))
    Ck = Ck ∪ Candidate;
}
if(s1.firstLevel = s2.lastLevel){
  Candidate = concat(s2, s1);
  if (!Prune(Candidate))
    Ck = Ck ∪ Candidate;
}

```

---

**Algorithm 6** Prune(s)

---

```

for all basic levels  $bl \in s$ ,
 $|bl| = m > 1$  {
  for all  $ist \subset bl$ ,  $|ist| = m - 1$  {
    seq = s with  $ist$  in-
stead of  $bl$ ;
    if seq  $\notin L_{k-1}$ 
      return true;
  }
}
if  $\nexists bl \in s, |bl| = m > 1$ 
  prefix = first  $k - 1$  levels;
  if prefix  $\notin L_{k-1}$ 
    return true;
  suffix = last  $k - 1$  levels;
  if suffix  $\notin L_{k-1}$ 
    return true;
return false;

```

---

candidate in the sequence. If the strings representing the itemsets are short enough we can use the XOR function to check containment fast. See Algorithm 8.

**Algorithm 7** SupportCount( $C_k$ )

---

```

for all sequences  $us \in T_D$ 
  for all candidates  $cand \in C_k$ 
     $cand.supp += Match(us, cand)$ ;
  if  $us$  the last sequence
    and  $cand.supp < min\_sup$ 
      remove  $cand$  from  $C_k$ ;

```

---

**3.4 Example**

Consider the dataset  $D$  from Figure 1. The transactions are sorted by  $userId$  and  $startDate$ . Given support of 30%, 6 basic events and all the target events are frequent. We enumerate the frequent basic events and map them to contiguous integers as illustrated in Figure 2. The same Figure presents an alternate representation of the target events. Figure 3 contains set of customer-sequences formed from  $D$ , which contain only the frequent events. Eight frequent pairs are presented in Figure 4 with their alternate representation. Note, that there are no frequent patterns of type  $U \rightarrow ev$ . Figure 5 illustrates all candidates received during the discovery process. The frequent patterns are in bold. During 4-sequences generation we prune such sequences as  $F \rightarrow \bar{U} \rightarrow AC$  (generated from  $F \rightarrow \bar{U}$  and  $\bar{U} \rightarrow AC$  by ConcatJoin),  $F \rightarrow \bar{U} \rightarrow A \rightarrow U$  (ConcatJoin on  $F \rightarrow \bar{U}$  and  $\bar{U} \rightarrow A \rightarrow U$ ),  $F \rightarrow \bar{U} \rightarrow C \rightarrow U$  (concat  $F \rightarrow \bar{U}$  with  $\bar{U} \rightarrow C \rightarrow U$ ),  $B \rightarrow \bar{U} \rightarrow E \rightarrow U$  ( $B \rightarrow \bar{U}$  and

**Algorithm 8** Match(us, cand)

---

```

count  $\leftarrow 0$ ;
for ( $i = 0; i < us.plevels.size -$ 
cand.plevels.size;
 $i++$ )
  for ( $j = 0; j <$ 
cand.plevels.size;  $j++$ ) {
     $cl = cand.level_j$ ;
     $sl = us.level_{i+j}$ ;
    if ( $(cl.basicLevel \subseteq sl.basicLevel$ 
or  $cl.basicLevel == \theta$ )
and  $cl.target == sl.target$ )
      count++;
    else break;
  }
return count;

```

---

**Frequent Basic Events**

event	den. by	sup	int	bit-string
(trendDay, increase)	$A$	50%	0	100000
(trendDay, flat)	$B$	40%	1	010000
(trendNight, increase)	$C$	50%	2	001000
(trendNight, flat)	$D$	30%	3	000100
(trendDayNight, flat)	$E$	40%	4	000010
(trendDayNight, decrease)	$F$	40%	5	000001

**Target events**

event	denoted by	support	bit
(DidUpgrade, Yes)	$U$	40%	1
(DidUpgrade, No)	$\bar{U}$	60%	0

**Figure 2. Enumeration and mapping the frequent events**

$\bar{U} \rightarrow E \rightarrow U$  accordingly) and  $F \rightarrow \bar{U} \rightarrow E \rightarrow U$ . The first two patterns are pruned since  $F \rightarrow \bar{U} \rightarrow A \notin L_3$ , the third is pruned because  $F \rightarrow \bar{U} \rightarrow C$  is not frequent, the fourth from the reason that  $B \rightarrow \bar{U} \rightarrow E \notin L_3$  and for the last  $F \rightarrow \bar{U} \rightarrow E \notin L_3$ . Some patterns were repeatedly generated, like  $\bar{U} \rightarrow A \rightarrow U$  and  $B \rightarrow \bar{U} \rightarrow A$  during the first step, and  $B \rightarrow \bar{U} \rightarrow AC$  and  $B \rightarrow \bar{U} \rightarrow A \rightarrow U$  during the second iteration of algorithm 5. In Figure 6 all target rules, given  $min\_conf$  70%, are presented. We also have four rules :  $A \rightarrow U$ ,  $C \rightarrow U$ ,  $\bar{U} \rightarrow A \rightarrow U$  and  $\bar{U} \rightarrow C \rightarrow U$  with confidence less than  $min\_conf$  (60%).

**4 The CSPADE algorithm**

In this section we describe the basic phases of slightly modified SPADE algorithm that we call CSPADE (Continuous Sequential Patterns Discovery using Equivalent

userId	startDate	endDate	trendDay	trendNight	trendDayNight	DidUpgrade
1	20/05/01	12/06/01	flat	flat	increase	No
1	13/06/01	09/09/01	increase	increase	increase	No
1	12/09/01	05/10/01	increase	increase	flat	Yes
2	25/01/01	03/02/01	decrease	decrease	decrease	No
2	10/02/01	30/01/01	increase	flat	flat	Yes
2	05/02/01	12/02/01	flat	decrease	decrease	No
2	13/02/01	20/02/01	increase	increase	decrease	Yes
3	30/01/01	15/02/01	flat	flat	flat	No
3	17/02/01	27/02/01	increase	increase	decrease	No
3	01/03/01	12/03/01	flat	increase	flat	Yes

Figure 1. Dataset D after sorting by userId and startDate

userId	sequence	alternate representation
1	$BD \rightarrow \bar{U} \rightarrow AC \rightarrow \bar{U} \rightarrow ACE \rightarrow U$	010100;0 → 101000;0 → 101010;1
2	$F \rightarrow \bar{U} \rightarrow ADE \rightarrow U \rightarrow BF \rightarrow \bar{U} \rightarrow ACF \rightarrow U$	000001;0 → 100110;1 → 010001;0 → 101001;1
3	$BDE \rightarrow \bar{U} \rightarrow ACF \rightarrow \bar{U} \rightarrow BCE \rightarrow U$	010110;0 → 101000;0 → 011010;1

Figure 3. Transformed dataset - set of customer sequences

pattern	support	alternate representation
$A \rightarrow U$	30%	100000;1
$C \rightarrow U$	30%	001000;1
$E \rightarrow U$	30%	000010;1
$B \rightarrow \bar{U}$	30%	010000;0
$F \rightarrow \bar{U}$	30%	000001;0
$\bar{U} \rightarrow A$	50%	NULL;0 → 100000;NULL
$\bar{U} \rightarrow C$	50%	NULL;0 → 001000;NULL
$\bar{U} \rightarrow E$	30%	NULL;0 → 000010;NULL

Figure 4. Frequent pairs

Classes). We made three main modifications to this algorithm:

- i. Using our definition of support,
- ii. We added an additional criterion for pruning during the processing of a class,
- iii. Filtering the non-continuous patterns after discovering all frequent sequences.

Recall that SPADE ([15]) works independently on equivalent classes which are specified by a common prefix. The main disadvantage of this algorithm is that partitioning into Equivalent Classes does not allow us to use our technique for generating only continuous patterns inside each class (the ConcatJoin and, in some cases, the ExpJoin do not work, because the seed sequences belong to different classes). Also, we cannot prune all non-appropriate patterns

candidate	support
$AC \rightarrow U$	20%
$AE \rightarrow U$	20%
$CE \rightarrow U$	20%
$BF \rightarrow \bar{U}$	10%
$\bar{U} \rightarrow AC$	<b>40%</b>
$\bar{U} \rightarrow AE$	20%
$\bar{U} \rightarrow CE$	20%
$\bar{U} \rightarrow A \rightarrow U$	<b>30%</b>
$\bar{U} \rightarrow C \rightarrow U$	<b>30%</b>
$\bar{U} \rightarrow E \rightarrow U$	<b>30%</b>
$B \rightarrow \bar{U} \rightarrow A$	<b>30%</b>
$B \rightarrow \bar{U} \rightarrow C$	<b>30%</b>
$B \rightarrow \bar{U} \rightarrow E$	0%
$F \rightarrow \bar{U} \rightarrow A$	20%
$F \rightarrow \bar{U} \rightarrow C$	20%
$F \rightarrow \bar{U} \rightarrow E$	20%

3-sequences

candidate	support
$\bar{U} \rightarrow AC \rightarrow U$	20%
$\bar{U} \rightarrow AE \rightarrow U$	20%
$\bar{U} \rightarrow CE \rightarrow U$	20%
$B \rightarrow \bar{U} \rightarrow AC$	<b>30%</b>
$B \rightarrow \bar{U} \rightarrow A \rightarrow U$	10%
$B \rightarrow \bar{U} \rightarrow C \rightarrow U$	10%

4-sequences

Figure 5. Candidates generation

rule	support	confidence
$E \rightarrow \bar{U}$	30%	75%
$B \rightarrow \bar{U}$	30%	75%
$F \rightarrow \bar{U}$	30%	75%
$\bar{U} \rightarrow E \rightarrow U$	30%	100%

Figure 6. Target Rules

during the class processing because they can produce appropriate candidates in the future (non-continuous patterns can form continuous). However, we can use the independence of classes and the fact, that each of them give patterns with the same prefix to limit the number of generated candidates. Recall that sequential patterns in our domain cannot have a “non-continuous” (or violated) structure, that is  $l_1 \rightarrow l_2 \rightarrow \dots \rightarrow l_k$ , where  $\exists i : l_i$  and  $l_{i+1}$  are both either target or basic levels, or  $l_i$  contains target event as well as basic ones. We avoid creation of classes specified by prefix that has such structure. This affects the way of creating new candidates, namely temporal join. We describe this later.

Note that the structure based method of pruning still does not guarantee that we’ll get only frequent continuous patterns. So, after processing of all classes we filter all patterns which are inappropriate to be target rules. We describe all the new phases below. Since the first, third and last phases, called Sorting the database, Transformaton phase and Rules generation respectively, are equivalent to the respective phases in Section 3, we omit these phases in the description. The phases describing partition into the Equivalent Classes and their processing do not differ from the respective phases of the original algorithm [15]. We briefly summarize them in the Appendix. The other phases are:

**2. The computation of the frequent 1-sequences** We use a vertical database format (see figure 8), where we maintain an id-list for each item. Each entry of the id-list is a  $\langle cid, tid \rangle$  pair where the item occurs (cid is the customer id and tid is the time id). Note, that for the basic event we associate the *start\_time* and for the target the *end\_time*. Using our definition of support, we do not calculate the fraction of customers supporting the pattern, but the number of occurrences of this pattern in the database. Given the vertical id-list database, all frequent 1-sequences can be computed in a single database scan. For each database item, we scan its id-list, incrementing the support for each new entry encountered.

**4. The computation of the frequent pairs** We use horizontal format of the database as described in the Appendix and [15]. We create all possible pairs, except for the Event Atoms (see Appendix) where at least one of the events is target. The reason is that such patterns will give non-

appropriate candidates in the future due to their illegal structure and temporal join properties. We give more explanations in the paragraph 4.

## 5. The decomposition into prefix-based parent equivalence classes

See Appendix and [15].

**6. Processing the classes** In addition to what is described in the Appendix and [15], we do not create new equivalent classes with a prefix which is a sequence with non-continuous structure, during the recursive application of  $\theta_k$ . We can prove that such class will give us only non-continuous patterns that will not affect the whole process of sequence generation due to the independence of classes. In order to avoid the creation of such classes we need to exclude the patterns forming them, namely, the patterns with violated prefix (denote them violated patterns). It affects the frequent sequence enumeration inside a class, namely the temporal join (see Appendix, 6.2). We add several checkings to the join 3. The candidates of joining  $P \rightarrow A$  and  $P \rightarrow B$  will be:  $P \rightarrow AB$  if it is valid,  $P \rightarrow A \rightarrow B$  if  $P \rightarrow A$  is valid and  $P \rightarrow B \rightarrow A$  if  $P \rightarrow B$  is valid.

It can be proved (we omit this proof here due to space limits), that these modifications in addition to constraints introduced in paragraph 4 guarantee, that we’ll never receive violated sequences during processing the classes.

We also use the support-based (see Appendix) pruning during processing of classes and the temporal join, but in calculation of the support we do not check the continuity of a pattern, and therefore we may receive a value which is equal or bigger than the actual support value. This is discussed next.

---

### Algorithm 9 Filtering of non-appropriate patterns

---

```

 $L_2 = \{p\}, p \in L_2$  and  $p$  is frequent
and continuous;
 $L \leftarrow L_2$ ;
 $k \leftarrow 3$ ;
while( $L_k \neq \emptyset$ ) {
  for all  $p \in L_k$  {
    for all  $subs \subset p, |subs| = |p| - 1$ ;
      if ( $subs \notin L$ ) {
        remove  $p$  from  $L_k$ ;
        break;
      }
  }
}
SupportCount( $L_k$ );
 $L = L_{k++}$ ;
}

```

---

cid	st_time	end_time	b_items	target
1	10	14	$C D$	$\bar{U}$
1	15	20	$A B C$	$U$
1	21	24	$A B F$	$\bar{U}$
1	25	31	$A C D F$	$U$
2	15	19	$A B F$	$\bar{U}$
2	20	25	$E$	$U$
3	10	15	$A B F$	$U$
4	10	15	$D G H$	$\bar{U}$
4	20	24	$B F$	$\bar{U}$
4	25	30	$A G H$	$U$

Figure 7. Original Input-Sequence Database

**7. Filtering the non-relevant sequences** As we already mentioned, after using the structure-based and support-based (see the Appendix) pruning techniques during processing of classes, we still may receive non-frequent sequences. Before checking support and continuity for the discovered patterns via scanning the customer-sequences, we save time by pruning potentially non-appropriate patterns. We use the pruning technique described in Section 3. It is clear that we cannot filter patterns of a class until we finish with all the classes (the subsequences of a pattern may belong to different classes). To start the checking, we filter  $F_2$  by scanning  $T_D$ , to form the seed set for the initial iteration of the Pruning procedure. Then we go from the next upper set  $L_3$  to the last one and, at each iteration  $i$ , filter the patterns of set  $L_{i-2}$  by generating appropriate subsequences for each pattern (see the Pruning subsection of Section 3), and check if they belong to the previous set  $L_{i-1}$ . If the pattern was not pruned, we scan  $T_D$  and check its frequency and continuity, using procedure Match, described earlier. The remaining patterns form the seed set for the next iteration. This is shown in Algorithm 9.

#### 4.1 Example

Consider the input database shown in Figure 7. The database has 8 basic events, 4 customers (specified by a cid), and 10 events in all. Figure 8 shows all the frequent events with a minimum support of 40% with their id-lists. The support of  $A$  is 60% and of  $B, F, U$  and  $\bar{U}$  is 50%. In Figure 9 the horizontal format for the events is depicted. All frequent pairs and 3-sequences, and lattice of equivalence classes, induced by them, are seen in Figure 14.

## 5 Performance evaluation and experiments

To compare the performance of our algorithms and represent their scale-up properties, we performed several experiments on a Dual Xeon workstation with a CPU clock

	CID	TID
A:	1	15
	1	21
	1	25
	2	15
	3	10
B:	4	25
	1	15
	1	21
	2	15
	3	10
F:	4	20
	1	21
	1	25
	2	15
	3	10
U:	4	20
	1	20
	1	31
	2	25
	3	15
$\bar{U}$ :	4	30
	1	14
	1	24
	2	19
	4	15
	4	24

Figure 8. Id-lists for the Atoms

dataset	$D$	$C$	$T$	$E$
DB1	102100	11540	9	20
DB2	102100	11540	9	11
DB3	81700	11530	7	20

Figure 10. Tested Datasets Parameters

cid	(item,tid) pairs
1	(A 15) (A 21) (A 25) (B 15) (B 21) (F 21) (F 25) (U 20) (U 31) ( $\bar{U}$ 14) ( $\bar{U}$ 24)
2	(A 15) (B 15) (F 15) (U 25) ( $\bar{U}$ 19)
3	(A 10) (B 10) (F 10) (U 15)
4	(A 25) (B 20) (F 20) (U 30) ( $\bar{U}$ 15) ( $\bar{U}$ 24)

Figure 9. Vertical-to-Horizontal Database Recovery

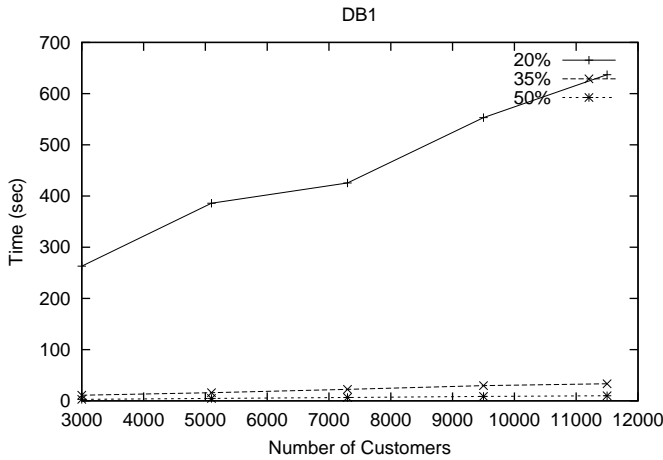


Figure 11. Scale-up of CTSPD

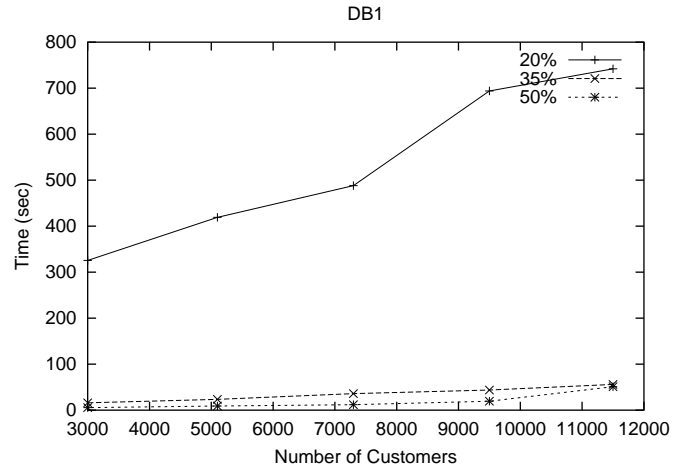


Figure 12. Scale-up of CSPADE

rate of 2.4 GHz, 2 GB RAM, running Linux. The data is stored on Oracle 9i. The algorithms were implemented in Java and intergrated with the FlexMine system[7].

### 5.1 The Tested Data

We evaluated the performance of the algorithms over relative big real-life data. We varies such parameters as number of customers in the dataset ( $C$ ), number of events per transaction ( $E$ ), average number of transactions per customer ( $T$ ) or number of transactions at all ( $|D|$ ). These parameters for the chosen 3 datasets (DB1, DB2 and DB3 respectively) are shown in Figure 10. DB3 is a 80% sample of DB1. In this paper we show the results of relative performance on DB1 and DB3.

### 5.2 Scale-up Properties

In this section we present the results of scale-up experiments for both our algorithms. We tested the scale-up both as depending on the number of customers and on the number of events in a single transaction. Figures 11 and 12 represent the scale-up experiments for both algorithms with the number of customers increased from 3000 to 11500. The results are shown for the DB1. We keep all parameters of dataset constant and fixed the minimum support, that

is 20%, 35% and 50% respectively. As shown, the execution time increased with the customers number increased. The CTSPD is faster and its execution time scale more uniformly. An execution time of CSPADE strongly increased on customer number equals 9500. We also made experiments with the number of events per transaction, that varies from 11 to 20, and with the number of generated rules, that varies from 1 to 482 in DB1, from 1 to 106 in DB2 and from 1 to 309 in DB3. Both algorithms show similar scale-up properties for both cases — an execution time increased with the number of events/generated rules increased. Also, we got experimental results, that demonstrate increasing the number of generated rules with the support decreased.

### 5.3 Relative Performance

Figure 13 presents the relative execution times for both algorithms. CSPADE is slower than CTSPD, because it generates a lot of candidates which are filtered at the end of process. The results are shown for DB1 with fixed minimum support, from 20% to 60%. As expected, the execution time increased as the minimum support decreased due to the increased number of generated rules. Also, we present the relative execution times as a function of the actual number of generated rules. These results received on DB3 and show that the execution time scales quite linearly.

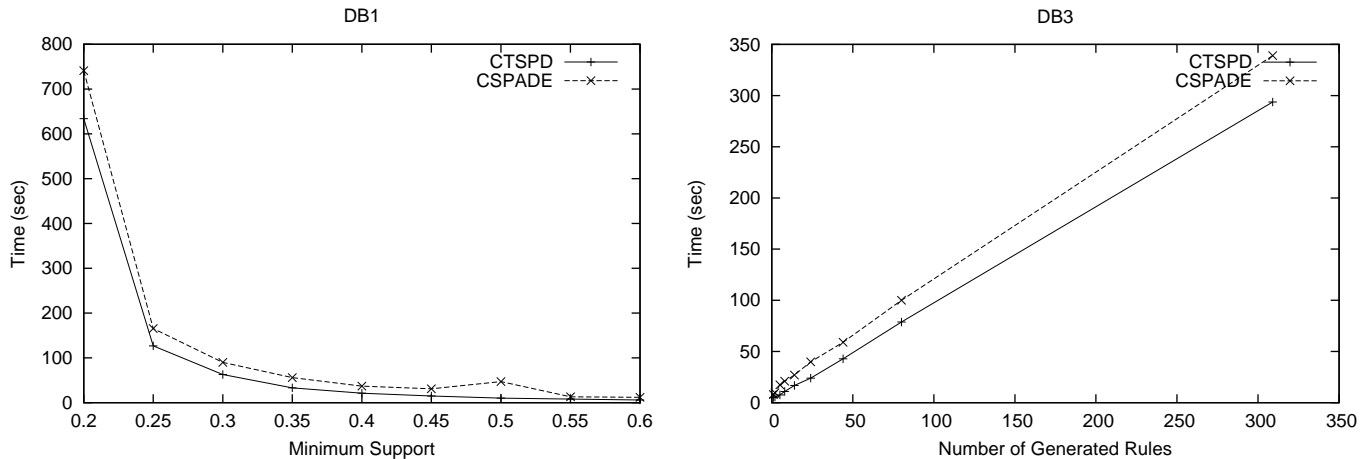


Figure 13. Relative Performance

## 6 Conclusions

In this paper a new problem definition was introduced for target rules mining, and two algorithms for solving it were proposed. Both algorithms, CTSPD and CSPADE, mine target sequential patterns, that have to be continuous, over the temporal database of customer transactions.

The first algorithm, CTSPD, is based on the Apriori principle, but adapted to our special application domain and considering only continuous candidates. The second algorithm, CSPADE, is based on SPADE and was adapted similarly to our continuous target based patterns.

We presented the relative performance and scale-up experiments for both algorithms, using real-life data. As expected, CTSPD was faster than CSPADE, which generated many more candidates. Both algorithms have comparable scale-up properties. The derived rules are used for improving the prediction of customers product upgrade that was introduced in [9]. After presenting the prediction experiments on our rules, we got results much better than any of those reported in [9].

## References

- [1] J. Adamo. *Data Mining for Association Rules and Sequential Patterns*. Springer-Verlag new York, 2001.
- [2] C. C. Aggarwal, C. M. Procopiuc, and P. S. Yu. Finding localized associations in market basket data. *Knowledge and Data Engineering*, 14(1):51–62, 2002.
- [3] R. Agrawal, T. Imieliski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216. ACM Press, 1993.
- [4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile, September 1994.
- [5] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of the Int'l Conference on Data Engineering (ICDE)*, Taipei, Taiwan, March 1995.
- [6] Antunes and Oliveira. Temporal data mining: an overview. In *KDD 2001 Workshop on Temporal Data Mining, 7th ACM SIGKDD*, San Francisco, CA, USA, August 2001.
- [7] R. Ben-Eliyahu-Zohary, C. Domshlak, E. Gudes, N. Liusternik, A. Meisels, T. Rosen, and S. E. Shimony. Fleximine - a flexible platform for kdd research and application development. *Annals of Mathematics and Artificial Intelligence*, 39(1-2):175–204, 2003.
- [8] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *Proceedings ACM SIGMOD*, pages 265–276, Tucson, Arizona, USA, May 1997.
- [9] A. Budker, E. Gudes, T. Hildeshaim, M. Litvak, E. Shimony, L. Amit, S. Meltzin, and G. Sotolorevsky. Targeting customers by mining usage time-series. In *CS/Stat'03 Second Haifa Winter Workshop on Computer Science and Statistics*, Haifa, Israel, December 2003.
- [10] Gary M. Weiss. Mining predictive patterns in sequences of events. In *Proc. of AAAI/GECCO Workshop on Data Mining with Evolutionary Algorithms: Research Directions*, 1999.
- [11] M. Gavrilov, D. Anguelov, P. Indyk, and R. Motwani. Mining the stock market: Which measure is best? (extended abstract).
- [12] H. Mannila, H. Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.
- [13] R. Srikant and R. Agrawal. Mining sequential patterns. generalizations and performance improvements. In *Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT)*, Avignon, France, March 1996.
- [14] J. T.-L. Wang, G. Chirn, T. Marr, B. Shapiro, D. Shasha, and K. Zhang. Combinatorial pattern discovery for scientific data: Some preliminary results. In *Proceedings of the 1994 ACM SIGMOD*, pages 115–125, Minneapolis, Minnesota, US, May 1994.

[15] M. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42(1, 2):31–60, January, February 2001. special issue on Unsupervised Learning (Doug Fisher, ed.).

## 7 Appendix

**4. The computation of the frequent pairs** We perform a vertical-to-horizontal transformation on-the-fly. This format consists of a list of  $\langle item, tid \rangle$  pairs for each cid (see figure 9). There are only frequent items present. Computing F2 from the recovered horizontal database is straightforward. We form a list of all 2-sequences in the list for each cid, counting number of occurrence of sequence for this cid, and update counts in a 2-dimensional array indexed by the frequent items.

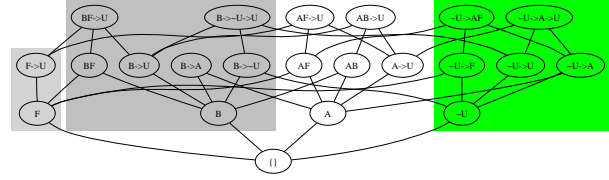
**5. The decomposition into prefix-based parent equivalence classes** We can represent all frequent sequences as a lattice, where the sequences are represented by the nodes and the edges between pair of sequences  $\langle s_1, s_2 \rangle$  mean that  $s_1$  is subsequence of  $s_2$ . If we had enough main-memory, we could enumerate all frequent sequences by traversing the lattice. However, we have a limited amount of main-memory and this brings up a question: can we decompose the original lattice into small pieces such that each piece can be solved independently in main-memory. The suggestion is to decompose the lattice to equivalence classes.

Define an equivalence relation  $\theta_k$  on the set of sequences as follows: two sequences are in the same class if they share a common  $k$  length prefix. We decompose all frequent 2-sequences into equivalence classes so that two sequences are in the same class if they share a common 1-length prefix - partition introduced by  $\theta_1$ . One can see such decomposition in Figure 14. Theoretically, we can decompose any  $F_k$  so, that  $k$ -sequences in the same class share a common  $k - 1$ -length prefix - by  $\theta_{k-1}$ .

**6. Processing the classes** We can process the classes by two methods: *DFS* or *BFS*. Both these methods are based on a recursive decomposition of each parent class into smaller classes induced by the equivalence relation  $\theta_k$ .

In *BFS* the lattice of equivalence classes is generated by the recursive application of  $\theta_k$  in a bottom-up manner. We process all the child classes at a level before moving on to the next level. For instance, we process the classes  $\{[AB], [AF], [A \rightarrow U]\}$  before moving on to the classes  $\{[AB \rightarrow U], [AF \rightarrow U]\}$ , and so on.

In *DFS*, we solve all child equivalence classes along one path before moving on to the next path. For example, we process the classes in the following order:  $[AB], [AB \rightarrow U], [AF], [AF \rightarrow U]$  and so on.



**Figure 14. Equivalence Classes Introduced by  $\theta_1$**

**6.1 Enumerating Frequent Sequences of a Class** Frequent sequences are generated by joining the id-lists of all pairs of atoms and checking the cardinality (number of  $\langle cid, time \rangle$  pairs) of the resulting id-list against  $min\_sup$ . Before joining the id-lists a pruning step can be inserted to ensure that all subsequences of the resulting sequences are frequent. If this is true, we can calculate the id-list join, otherwise we can avoid the temporal join. The sequences found to be frequent at the current level form the atoms of classes for the next level. This recursive process is repeated until all frequent sequences have been enumerated.

**6.2 Temporal Id-List Join** We now describe how we perform the id-list joins for two sequences. Consider an Equivalence class  $[B \rightarrow A]$  with the atom set  $\{B \rightarrow AB, B \rightarrow AD, B \rightarrow A \rightarrow A, B \rightarrow A \rightarrow D, B \rightarrow A \rightarrow F\}$ . If we let  $P$  stand for the prefix  $B \rightarrow A$ , then we can rewrite the class to get  $[P] = \{PB, PD, P \rightarrow A, P \rightarrow D, P \rightarrow F\}$ . The class has two kinds of atoms: the event atoms  $\{PB, PD\}$ , and the sequence atoms  $\{P \rightarrow A, P \rightarrow D, P \rightarrow F\}$ . To extend the class it is sufficient to join the id-lists of all pairs of atoms. However, depending on the atom pairs being joined, there can be up to 3 possible resulting frequent sequences:

1. *Event Atom with Event Atom*: If we are joining  $PB$  with  $PD$ , then the only possible outcome is new event atom  $PBD$ .
2. *Event Atom with Sequence Atom*: If we are joining  $PB$  with  $P \rightarrow A$ , then the only possible outcome is new sequence atom  $PB \rightarrow A$ .
3. *Sequence Atom with Sequence Atom*: If we are joining  $P \rightarrow A$  with  $P \rightarrow F$ , then there are three possible outcomes: a new event atom  $P \rightarrow AF$ , and two new sequence atoms  $P \rightarrow A \rightarrow F$  and  $P \rightarrow F \rightarrow A$ . A special case arises when we join  $P \rightarrow A$  with itself, which can produce only the new sequence atom  $P \rightarrow A \rightarrow A$ .

# Incremental Info-Fuzzy Algorithm for Real Time Data Mining of Non-Stationary Data Streams

Lior Cohen

Gil Avrahami

Mark Last

*Ben-Gurion University of the Negev  
Department of Information Systems Engineering  
Beer-Sheva 84105, Israel  
Email: {clior, gilav, mlast}@bgu.ac.il*

## Abstract

*Most real-world data streams are generated by non-stationary processes that may change drastically over time. In our previous work, we have presented a real-time data mining algorithm called OLIN (On-Line Information Network), which adapts itself automatically to the rate of concept drift in a non-stationary data stream by repeatedly constructing a new model from a sliding window of latest examples. In this paper, we introduce an incremental version of the OLIN algorithm, which saves a significant amount of computational effort by updating an existing model as long as no concept drift is detected. The approach is evaluated on large real-world streams of traffic and stock data.*

## Keywords

Real-time data mining, incremental learning, online learning, concept drift, info-fuzzy networks.

## 1. Introduction

In the last 20 years, there has been a huge increase in the amount of information and data, which are kept in databases and computer files. The data to be stored is doubling itself every year. Study made by the UC Berkeley's School of Information Management and Systems in 2003 about the storage and flows analyzes of data [3], indicates the following findings:

- Information flow via electronic channels - telephone, radio, TV, and the Internet (The World Wide Web contains about 170 terabytes of information), contained almost 18 exabytes of new information. Three and a half times more than is recorded in storage media.
- Each year, 800 MB of recorded information is produced per person.
- Ninety-two percent of new information is stored on magnetic media, primarily hard disks.

According to another study [12], transaction-processing workload climbed from an average of 2,094 transactions per second (tps) in 2001 to 3,223 tps two years later, an increase of 54 percent.

The modern information technology produces every year more powerful computers, which enable us to collect, store, transfer, and combine huge amounts of data at very low cost. The progress of the digital data acquisition and technology of storage made the growth of the huge databases possible. This progress in database capability has influenced many areas in human's life, from supermarket transaction data and credit card usage records to molecular and medical databases.

The growth in data has also increased the difficulties and challenges of extracting valid and potentially useful information from these databases, which is the main task of data mining and Knowledge Discovery in Databases (KDD). The main difficulty in mining non-stationary data streams is to cope with the changing data concept. The fundamental processes generating most real-time data streams may change over years, months and even seconds, at times drastically. This change, also known as concept drift [1], causes the data-mining model generated from past data, to become less accurate in the classification of new data. Algorithms and methods, which extract patterns from continuous data streams, are known as online learning [4]. Real-time data mining of high-speed data streams has large potential in fields such as monitoring manufacturing processes, prediction of stock prices, and intrusion detection in computer networks.

In this paper, we propose a new, incremental approach to mining non-stationary data streams. The main idea of the incremental approach is to increase the average classification rate (in records per second) of real-time data mining systems by reducing the number of times a completely new model is generated. The proposed approach is evaluated on an incremental version of the On-Line Information Network (OLIN) algorithm presented by Last in [5]. OLIN is based on the batch Info-Fuzzy Network (IFN) algorithm developed by Last & Maimon [6] [7]. The proposed incremental algorithm keeps updating an

existing model as long as no concept drift is detected in the arriving data. While most of the existing online algorithms, which deal with the problem of concept drift, build a new model from every new window of training examples, our incremental approach saves the expensive CPU time by performing minimal changes in the current structure of the classification model.

## 2. Related Work

When dealing with non-stationary data streams, the optimal situation is to have KDD systems that operate continuously, constantly processing the data received so that potentially valuable information is never lost. In order to achieve this goal, several methods for extracting patterns from non-stationary streams of data have been developed, all under the general title of online (incremental) learning methods.

The pure incremental learning methods take into account every new instance that arrives. These algorithms may be irrelevant when dealing with high-speed data streams. Widmer & Kubat [2] have described a series of purely incremental learning algorithms that flexibly react to concept drift and can take advantage of situations where context repeats itself. The series of algorithms is based on a framework called FLORA. FLORA maintains a dynamically adjustable window during the learning process and whenever a concept drift seems to occur (a drop in predictive accuracy) the window shrinks (forgets old instances), and when the concept seems to be stable the window is kept fixed. Otherwise, the window keeps growing until the concept seems to be stable. FLORA is a computationally expensive methodology, since it updates the classification model with *every* example added to or removed from the training window.

Domingos & Hulten [8] have proposed the VFDT (Very Fast Decision Trees learner) system in order to overcome the longer training time issue of the pure incremental algorithms. The VFDT system is based on a decision tree learning method, which builds the trees based on sub-sampling of a stationary data stream. To deal with changing data streams, Domingos & Hulten [9] have proposed an improvement to the VFDT algorithm which is the CVFDT (Concept-adapting Very Fast Decision Tree learner). CVFDT applies the VFDT algorithm to a sliding window of a fixed size and builds the model in an incremental manner instead of building it from scratch whenever a new set of examples arrives. CVFDT increases the computational overload vs. VFDT by growing alternate sub-trees at its internal nodes. The model is modified when the alternate becomes more accurate than the original.

Last in [5] describes an online classification system that uses an info-fuzzy network (IFN). The system called OLIN (On Line Information Network) gets a continuous stream of non-stationary data and builds a network based on the

latest examples (sliding window). OLIN detects a concept drift (an unexpected rise in the classification error rate) and dynamically adjusts the size of the training window and accordingly, the rate of the model reconstruction. The calculations of the window size in OLIN are based on the information theory and statistics. The experimental results of [5] show that in non-stationary data streams, dynamic windowing generates more accurate models than the static (fixed size) windowing approach used by CVFDT.

## 3. The IFN Algorithm

Many learning methods use information theory to induce classification rules. One of the methods, developed by Last & Maimon [6] [7] is the IFN algorithm. IFN, or Info-Fuzzy Network, is an oblivious tree-like classification model, which is designed to minimize the total number of predicting attributes. The underlying principle of the IFN method is to construct a multi-layered network in order to test the Mutual Information (MI) between the input and output attributes. Each hidden layer is related to a specific input attribute and represents the interaction between this input attribute and the other ones. The IFN algorithm is using the pre-pruning strategy: a node is split if this procedure brings about a statistically significant decrease in the entropy value (or increase in the mutual information) of the target attribute. If none of the remaining input attributes provides a statistically significant increase in mutual information, the network construction stops. The output of this algorithm is a network, which can be used to predict the values of a target attribute similarly to the prediction technique used in decision trees.

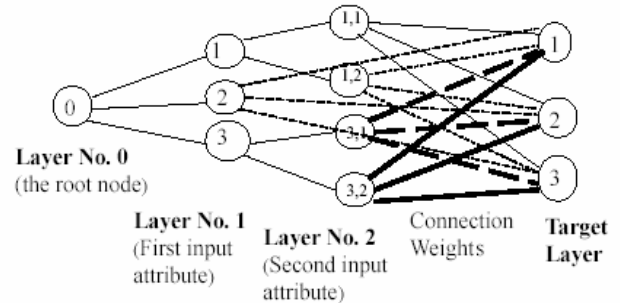


Figure 1. Info-fuzzy Network Two Layered Structure [7]

Fig. 1 illustrates a sample structure of an info-fuzzy network. In this example, the network contains two layers, which represent two input attributes. The first input attribute has three values, represented by nodes no. 1, 2, and 3 in the first layer. Nodes no. 1 and 3 are split by the network construction procedure. The second layer has four nodes, which are the combinations of two values of the second input attribute with two split nodes of the first layer.

The target layer represents the target attribute, which gets three values.

## 4. Incremental On-Line Information Network

### 4.1. Algorithm Overview

The online algorithm of Last [5] deals with the concept drift problem simply by generating a new model for every new sliding window. On one hand, this regenerative approach ensures accurate and relevant models over time and therefore an increase in the classification accuracy. On the other hand, the OLIN algorithm has a major drawback, which is the high cost of generating new models. The regenerative on-line IFN does not take into account the costs involved in replacing the existing model with a new one.

In this paper, we present a new algorithm, called Incremental On-Line Information Network (Incremental OLIN), which is an extension, to the regenerative OLIN algorithm of Last [5]. As shown in the evaluation section below, the incremental algorithm achieves almost the same and sometimes even higher accuracy rates than the regenerative algorithm and it is significantly cheaper since it does not require producing a new model for every new window of examples. As long as no concept drift is detected, the Incremental OLIN algorithm is applied repeatedly to a sliding window in order to update the current model rather than replace it completely.

The basic intuition behind the incremental approach is to update the current classification model with the current training window concept as long as no major concept drift has been detected and to build a new model in case of a major concept drift. From applying the regenerative OLIN [5] to several data sets (in transportation, manufacturing, and stock market domains) a few phenomena have been observed. The first one: when there is no concept drift between two adjacent training windows, the differences between the new constructed model and the former one are minor (about 80% of the network structure remains the same). This means that instead of re-constructing a new model (in case of no concept drift), the latest model can be updated in much less time and effort. The second one: when there is no concept drift between two adjacent windows, the main differences between the new model and the former model are in the last hidden layer. The third one: when a concept drift is discovered between two adjacent windows, the new model is almost totally different from the former one (in 90% of the cases the differences propagate up to the root node). The above phenomena indicate that in updating an existing model, it is sufficient to update its last layer and that if a concept drift has been detected, it is preferable to construct a new model.

Several operations for updating the model can be applied. One of them is to check the split validity on each

node starting from the root and downwards the network. This check is made in order to ensure that the current split of a specific node actually contributes to the mutual information calculated from the current training set. The elimination of non-relevant splits should decrease the error rate. Another operation is to check which attribute is more appropriate to correspond to the last (terminal) layer: the attribute that is already associated with the last layer or the second best attribute for the last layer of the previous model. The attribute selected for the last layer of the new model will be the one with the greater conditional mutual information based on the current training window. The last operation is attempting to split the nodes of the last layer on attributes, which are not yet participating in the updated network.

### 4.2. Detailed Description

Following is the pseudo-code outline of the Incremental OLIN algorithm.

The *IFN\_Control* procedure is responsible for managing the application. It gets as input a continuous stream of examples ( $S$ ). The initial size of the training window  $W_{init}$  is calculated by using an equation developed by Last in [5]. Afterwards, the initial IFN model is produced by applying the IN algorithm to the initial window of examples.

---

#### *IFN\_Control* ( $S$ )

Calculate the initial size of the training window  $W_{init}$

Set  $W = W_{init}$

Obtain IFN model by applying the IN algorithm to the sliding window ( $W$ )

While  $S$  is not finished

$IFN\ model = Incremental\_IFN(W, IFN\ model)$

Return  $IFN\ model$

---

The algorithm will run till the end of the data stream. If the data stream is infinite, the algorithm will keep running and producing IFN models for classification.

The *Incremental\_IFN* procedure is responsible for calculating both error rates of the training and validation examples after running the current model on those data sets. In addition, the maximum expected difference between those errors  $Max\_Diff$  is calculated at the 99% confidence level using a Normal Approximation to the Binominal distribution (as shown in [5]). It is important to mention that we assume immediate availability of correct classifications for the window of validation examples. This is a reasonable assumption in stock prediction, traffic control, web usage mining, and other real-time data mining domains [2], [5].

A stable concept is observed if the actual difference between the validation and training errors is smaller than the maximum expected difference  $Max\_Diff$ . In this case,

operations for updating the network are applied. If a concept drift occurs, we are forced to create a new network.

---

**Incremental\_IFN ( $W$ , IFN model)**

Calculate the training error rate  $E_{tr}$  of IFN model  
 Calculate the validation error rate  $E_{val}$  of IFN model  
 Find the maximum expected difference between the last training and the validation errors  $Max\_Diff$   
 If  $(E_{val} - E_{tr}) < Max\_Diff$  //concept is stable  
     Update\_Current\_Network (IFN\_Model,  $W$ )  
 Else  
     Obtain new IFN model by applying the IN algorithm to the sliding window ( $W$ )  
 Calculate New\_Training\_Window\_Size ( $W$ ) [5]  
 Return updated IFN\_Model

---

The **Update\_Current\_Network** procedure gets as inputs the current network structure and a sliding window. This procedure activates another procedure (**Check\_Split\_Validity**) for checking the split validity of the current network. Afterwards, it replaces the last layer of the network if needed. Finally, it activates the **New\_Split\_Process** procedure performing a new split process on the last layer (whether it was replaced or not).

---

**Update\_Current\_Network (IFN\_Model,  $W$ )**

Check\_Split\_Validity (IFN\_Model,  $W$ ) on the last layer of the current model  
 Calculate the conditional MI of  $Sec\_Best\_Attr$  based on the current training set ( $W$ )  
 IF (conditional MI of the current last layer < conditional MI of  $Sec\_Best\_Attr$ )  
     Replace last layer with  $Sec\_Best\_Attr$   
     New\_Split\_Process (IFN) on the last layer of the current model

---

**Check\_Split\_Validity** is responsible to check if the current split of each node actually contributes to the conditional mutual information calculated from the current training set.

---

**Check\_Split\_Validity (IFN\_Model,  $W$ )**

For  $i = total\_number\_of\_layers - 1$  to  $i = 1$   
 For  $j = 1$  to  $j = number\_of\_nodes\_in\_hidden\_layer\ i$   
     If node  $j$  is split  
         Calculate the estimated conditional MI of  $j$  and the target attribute [6]  
         Calculate the Likelihood-ratio statistic of  $j$  [6]  
         If the Likelihood-ratio statistic of  $j$  is significant  
             Leave the node split  
         Else  
             Remove the splitting and make  $j$  a terminal node

---

**New\_Split\_Process** is responsible for splitting the nodes of the last layer on attributes, which are not yet included in the updated network.

---

**New\_Split\_Process (IFN)**

Repeat for every candidate input attribute  $i'$  which is still not an input attribute  
     Repeat for every node  $z$  of the final hidden layer  
         Calculate the estimated conditional MI of  $i'$  and the target attribute given  $z$   
         Calculate the Likelihood-ratio statistic of  $i'$  and the target attribute given  $z$   
         If the Likelihood-ratio statistic of  $i'$  is significant  
             Split  $z$  on  $i'$  and increment the conditional MI of the candidate input attribute  $i'$  and the target

---

The time complexity of the suggested incremental algorithm depends on the number of detected concept drifts. As long as no concept drift has been detected, the major computational cost is to add a new layer to the existing network. Checking the split validity of the nodes in the network is relatively cheap ( $O(n)$  where  $n$  is the number of nodes in the network). In case of concept drift, a new network should be constructed from scratch, and the cost of the network construction procedure is linear in the number of records, linear in the number of distinct attribute values, and quadratic in the number of candidate input attributes [6].

## 5. Evaluation

The Regenerative OLIN reconstructs a new model with every new sliding window of training examples. According to the results presented [5], the classification models produced by this algorithm are relatively accurate but the total processing time is very long due to high frequency of constructing a new model. The Incremental OLIN algorithm presented in this paper is aimed at decreasing the processing time per each new example. The Incremental OLIN was evaluated in comparison to the original Regenerative OLIN on several real-world data streams.

### 5.1. Traffic Data

The first set of data streams includes traffic flow information on a signaled three-way intersection in Jerusalem. The traffic data is recorded by lane sensors. The vehicles can cross the intersection in five different directions. The five data streams obtained for directions 1 to 5 included the hourly incoming traffic volumes for 24 hours a day, seven days a week during a period of more than 3 years.

Each original data stream has been converted into a data set, where a record contains twelve candidate attributes representing the exact time (date, hour, day in week, etc.)

when the traffic volume was measured and traffic volumes at earlier points of time (the previous hour, the same hour of the previous day, etc.). The target attribute represented the volume of traffic during a given hour. Due to the fact that the target attribute is continuous we have manually discretized it to three intervals of high, medium, and low traffic volume. The traffic data was divided into five separate data tables for the five directions. Each table corresponding to a given direction contained about 30,000 hourly records.

### 5.2. Stock Data

The second set we used was a stock market data. This data set was also used in [5] for the evaluation of the Regenerative OLIN. The raw data represents the daily stock prices of 373 companies from the Standard & Poor's index [10], over a 5-year period (from 8/29/94 to 8/27/99). The data was obtained from the Microsoft™MoneyCentral web site [11]. An average of 15.64 intervals (with distinct trends) per company have been identified and the classification problem has been defined as predicting the correct length of the current interval based on the known characteristics of the current and the preceding intervals. The data table contains 5,462 records with six candidate attributes, which include the duration, the slope and the fluctuation measured in each interval as well as the sector of the corresponding stock. The target attribute which is the

duration of the second interval in an interval-pair, has been discretized to five intervals of nearly equal frequency.

### 5.3. Initial Results

The runs of both algorithms were carried out on a Pentium IV processor with 256 MB of RAM. In the experiments, the online learning on the traffic data starts after inducing the initial model from the first 500 records, which leaves the system to work with about 30,000 records for each direction. For the stock data, the online learning starts after the first 462 records, which are used for inducing the initial model. Tables 1 and 2 show the results after applying the regenerative and the Incremental OLIN to the traffic data sets and the stock data set respectively.

From the results on the traffic data, it can be seen that the Incremental OLIN has reduced the run time by at least 20% and at most by 87% (the average run time saving was 75%). At the same time, the accuracy rate decreased by 4% in the worst case while the average accuracy loss was only 2%. One can also see that the incremental algorithm has increased the classification rate from 55 to 139 records per second. In the case of the stock data, the incremental version reduced the run time by 72.25% while increasing the error rate by 1.3% only. In addition, the classification rate increased from 47.97 to 172.85 records per second, when we used the incremental version.

**Table 1. Summary of Experiments Using the Regenerative OLIN**

	Total Number of Records	Run Time (sec.)	Average Classification Rate (records/second)	Error Rate	Number of Observed Concept Drifts
Direction1	28,761	487.68	58.975	0.107	55
Direction2	30,480	466.38	65.354	0.215	55
Direction3	30,480	1354.42	22.5	0.103	17
Direction4	30,480	1829.57	16.66	0.114	7
Direction5	30,480	271.96	112.075	0.111	51
Traffic Average	30,136	882	55.11	0.13	37
Stock Market	5462	113.87	47.97	0.415	8

**Table 2. Summary of Experiments Using the Incremental OLIN**

	Total Number of Records	Run Time (sec.)	Average Classification Rate (records/second)	Error Rate	Number of Observed Concept Drifts
Direction1	28,761	222.3	129.38	0.122	79
Direction2	30,480	211.69	143.98	0.256	119
Direction3	30,480	192.23	158.56	0.103	20
Direction4	30,480	242.12	125.89	0.145	25
Direction5	30,480	217.23	140.31	0.119	65
Traffic Average	30,136	217.11	139.62	0.149	61.6
Stock Market	5462	31.6	172.85	0.428	11

## 6. Conclusions and Future Work

This paper has presented a new, incremental approach to real-time classification of continuous non-stationary data streams. The incremental approach applies the classification algorithm repeatedly to a sliding window of examples, in order to update the existing model or to construct a new model. A concept drift is detected by an unexpected rise in the classification error rate. When the concept appears to be stable, the system updates the current classification model by applying several operations to it. If a concept drift has been detected, the system re-generates a new model.

The proposed incremental approach was evaluated using an incremental version of the On-Line Information Network (OLIN) algorithm, which constructs an oblivious tree-like classification model. The efficiency of the Incremental OLIN has been confirmed by experiments on real-world sets of online data. The data in use included a set of five traffic data streams, which contained about 30,000 records for each traffic direction and a set of stock data, which contained 5,462 records. It is clear that the incremental version of OLIN outperforms the Regenerative OLIN in terms of processing rate. Another encouraging finding was the low decrease in accuracy of the incremental version compared to the regenerative one.

The future work includes implementation of additional incremental classifiers and evaluating them on more real-world datasets, as well as on artificially built data streams. Finding a better trade-off between the accuracy rate and the processing time can also be examined.

**Acknowledgments.** We would like to thank the Traffic Control Center of Jerusalem for granting us the permission to use their traffic database. This work was partially supported under a research contract from the Israel Ministry of Defense.

## 7. References

- [1] D.P Helmbold and P.M. Long, Tracking Drifting Concepts by Minimizing Disagreements, *Machine Learning*, No. 14, pp. 27-45, 1994.
- [2] G. Widmer and M. Kubat, "Learning in the Presence of Concept Drift and Hidden Contexts", *Machine Learning*, Vol. 23, No. 1, pp. 69-101, 1996.
- [3] P. Lyman and H. R. Varian, "How Much Information", 2003. Retrieved from <http://www.sims.berkeley.edu/how-much-info-2003>.
- [4] M. Black and R. J. Hickey, "Maintaining the Performance of a Learned Classifier under Concept Drift", *Intelligent Data Analysis*, No. 3, pp. 453-474, 1999.
- [5] M. Last, "Online Classification of Nonstationary Data Streams", *Intelligent Data Analysis*, Vol. 6, No. 2, pp. 129-147, 2002.
- [6] M. Last and O. Maimon, "A Compact and Accurate Model for Classification", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, No. 2, pp. 203-215, February 2004.
- [7] O. Maimon and M. Last, *Knowledge Discovery and Data Mining - The Info-Fuzzy Network (IFN) Methodology*, Kluwer Academic Publishers, December 2000.
- [8] P. Domingos and G. Hulten, "Mining High-Speed Data Streams", *Proc. of KDD 2000*, pp. 71-80, 2000.
- [9] P. Domingos and G. Hulten, "Mining Time-Changing Data Streams", *Proc. of KDD 2001*, pp. 97-106, ACM Press, 2001.
- [10] Standard & Poor's Index at <http://www.spglobal.com>.
- [11] The Microsoft<sup>TM</sup> MoneyCentral home page at <http://windowsmedia.com/mediaguide/gbhome>.
- [12] R. Winter and K. Auerbach, "Contents Under Pressure", *Intelligent Enterprise*, May 2004, available at <http://www.intelligententerprise.com/showArticle.jhtml?articleID=18902161>

# Learning from data streams via online transduction

Shen-Shyang Ho and Harry Wechsler

Department of Computer Science  
George Mason University  
4400 University Drive  
MSN 4A5  
Fairfax, VA 22030

E-mail: {sho, wechsler}@cs.gmu.edu

## Abstract

*A practical issue in the existing transduction methods is expensive and inefficient computation compared to induction methods. This has hindered the use of transduction methods in temporal and real-time data mining.*

*In this paper, we introduce a fast incremental transductive confidence machine (TCM) based on adiabatic incremental support vector machine (SVM) such that critical information from current transduction trial is stored for later use. The algorithm is empirically shown to be computationally efficient and its performance is consistent with standard TCM implementation.*

*Besides being a classifier, TCM provides additional useful statistical information about the data that it processed. These information can be useful for temporal and real-time data mining. We demonstrate the feasibility and usefulness of using such statistical information for stream-based active learning.*

## 1. Introduction

A simple interpretation of transduction is that the unknown values at some points of interest are estimated *directly* from the training data [15]. This is different from induction when a general rule is inferred from the training data and predictions are based on this general rule. The practical distinction between transduction and induction is whether we extract and store the general rule or not [18]. Current transduction methods are computationally expensive such that any improvement in performance cannot justify this expensive computation. Hence, transduction methods are seldom or never taken seriously as viable temporal or real-time data mining techniques.

One such transduction method, called transductive confidence machine (TCM), proposed by Gammernan and his colleagues [2] [3], predicts the classification of an object together with a measure of confidence using the Algorithmic Randomness Theory in an off-line setting. Their first proposed algorithm is based on support vector machines (SVM). Vovk [17] proved later that TCM, independent of the base classifier, is well-calibrated (in the sense that in the long run, the number of wrong predictions is bounded) in an online setting with i.i.d. assumption.

Transduction methods, in general and TCM in particular, are computationally expensive and are practical only for small data sets. To improve the computational speed, Saunders et. al. [13] used a hashing function to split the training set into smaller subsets of roughly equal size, which are used to construct a number of support vector machines in the TCM implementation. This solution, however, assumes the availability of all the training examples, is not feasible in an on-line setting. A modification of TCM, called inductive confidence machine (ICM) [10], is also proposed and sacrifices some prediction accuracy for efficiency.

In this paper, we introduce a fast incremental TCM that integrates the incremental SVM by *adiabatic increment*, i.e. preserving Karush-Kuhn-Tucker conditions on all previously seen training examples with the addition of a new example [1], into TCM. Our implementation, without splitting up the current training set into subsets and without sacrificing prediction accuracy, significantly improves the computational efficiency of TCM.

Besides being a classifier, TCM provides additional useful statistical information about the data that it processed. We show the feasibility and usefulness of using such statistical information for stream-based active learning.

In Section 2, we review some fundamental concepts in algorithmic randomness and p-value for understanding of

TCM. Section 3 describes the on-line (randomized) TCM and the idea of region predictor. In Section 4, we review different implementation strategies for incremental support vector machine and in Section 5, we briefly review the adiabatic increment for incremental SVM. In Section 6, we discuss the difference between online and incremental learning. In Section 7, we discuss using an incremental classifier to improve the computational efficiency of transduction and then we introduce our online incremental TCM algorithm. In Section 8, we perform experiments to compare the computational efficiency and performance of our algorithm with the standard TCM implementation and an implementation of TCM using a different incremental strategy. We also demonstrate the application of our incremental TCM on stream-based active learning. Throughout the rest of the paper, we assume basic familiarity of SVMs [15].

## 2. Randomness, p-value and Transductive Confidence Machine

Vovk and et. al. [16] first proposed the application of algorithmic randomness on machine learning problems. The confidence measure used in TCM is based upon universal tests for randomness, or their approximation. A Martin-Löf randomness deficiency [8] based on such tests is a universal version of the standard statistical notion of p-values.

By reformulating the Martin-Löf randomness test, we have a function  $t : Z^* \rightarrow [0, 1]$  called a *p-value function* if

1. Let  $P_n$  be the set of probability distribution in  $Z^n$ . For all  $n \in \mathcal{N}$  and  $r \in [0, 1]$  and all  $P \in P_n$ ,

$$P\{z \in Z^n : t(z) \leq r\} \leq r \quad (1)$$

2.  $t$  is semi-computable from above, i.e. there exists a computable sequence of computable functions  $t_i : Z^* \rightarrow [0, 1], i = 1, 2, \dots$  such that  $t(z) = \inf_i t_i(z)$  for all  $z \in Z^*$ .

where  $Z$  is the set of all possible labeled examples,  $Z^n$  is the set of all sequences of  $n$  labeled examples of the form

$$\{z_1, z_2, \dots, z_n\} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

where  $x_i$  is an object and  $y_i$  is its label for  $i = 1, 2, \dots, n$  and  $Z^*$  is the set of all finite sequences of labeled examples [11]. This p-value function is practically equivalent to the standard statistical notion of p-values. A particular p-value for a sequence of labeled examples is also called the *randomness level* of that sequence [3].

In the literature on significance testing, the p-value is often defined as the probability of observing a point in the sample space which can be considered as extreme as, or

more extreme than, the observed sample [19]. This calculation requires a well defined stochastic ordering of the sample space which can be provided by a test statistic. The p-value serves as a measure of how well the data support or discredit a null hypothesis: the smaller the p-value, the greater the evidence against the null hypothesis. In other words, smaller p-values of the observed level of significance favor the alternative hypothesis and larger values favor the null hypothesis.

In order to construct a valid p-value function, the main component of a TCM, an ordering function called *strangeness measure* [3] (or nonconformity score [18]) is required. The strangeness,  $\alpha_i$ , of a particular labeled example,  $z_i$ , corresponds to the uncertainty of that example with respect to all other labeled examples: the higher the measure, the higher the uncertainty. The strangeness measure used depends on the type of base classifier used to construct a TCM. To use support vector machine (SVM) as the base classifier to construct a TCM, the solution, a set of Lagrange multipliers, of the dual problem of the SVM optimization problem is used as the strangeness measure [2]. A strangeness measure for k-nearest neighbor is given in [11].

Consider a sequence of labeled examples,  $\{z_1, z_2, \dots, z_{n-1}\}$  with their corresponding strangeness values,  $\alpha_1, \alpha_2, \dots, \alpha_{n-1}$  and an unlabeled example,  $z_n$ , assigned a particular label and its strangeness value,  $\alpha_n$ , we define a p-value function  $t : Z^n \rightarrow [0, 1]$  which returns a p-value of  $z_n$  (assigned the particular label) by

$$t(z_1, z_2, \dots, z_n) = \frac{\#\{i = 1, \dots, n : \alpha_i \geq \alpha_n\}}{n} \quad (2)$$

which satisfies

$$P\{(z_1, z_2, \dots, z_n) : t(z_1, z_2, \dots, z_n) \leq r\} \leq r$$

for any  $r \in [0, 1]$  and for any probability  $P$  that the probability distribution of  $Z$  induced over the choice of  $n$  examples. The strangeness value of an example in the sequence must be independent of its position in the sequence for the p-value function to be valid. This p-value computation is a simple approximation of the randomness level of the Martin-Löf's randomness test.

Below is the algorithm for the approximation of p-values for all possible labels that an unlabeled example can be assigned. This algorithm is the essential procedure of a TCM:

### Algorithm:

Input: training set  $T = \{z_1, z_2, \dots, z_{n-1}\}$  and an unlabeled example  $z_n = (x_n, ?)$ :

1. FOR  $y = 1$  to  $c$  (number of possible classes)
  - (a) Label  $z_n$  as class  $y$
  - (b) Construct a classifier (e.g. *SVM*) using  $T$  and  $(x_n, y)$

- (c) Use *SVM* to compute strangeness,  
 $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$  for  $T$  and  $(x_n, y)$
- (d) Use strangeness,  $\alpha$  to compute p-value for  
 $(x_n, y)$  (Equation (2))

ENDFOR

2. Assign the label with the largest p-value to  $z_n = (x_n, ?)$
3. Confidence = 1 - second highest p-value
4. Credibility = highest p-value

### 3. Online Transductive Confidence Machine

The basic idea of online TCM is similar to the above procedure. A randomized version of TCM called *randomized Transductive Confidence Machine (rTCM)* introduced by Vovk [17] is used here so that a prediction error bound for online TCM can be achieved. The computation of p-value for TCM and its randomized version is identical.

TCM is a way to define a *region predictor* [17] (or conformal predictor [18]) from a learning algorithm that can only return a prediction for each unlabeled test input. A region predictor is a function when given an unlabeled test input and a confidence level returns a set of possible predictions with a degree of confidence.

For a particular unlabeled example  $z_n = (x_n, ?)$ , a training set  $T$ , a set of all possible labels  $Y$  and  $\delta \in (0, 1)$ , we can obtain a region predictor:

$$\Gamma_{1-\delta}(T, z_n)$$

which contains  $y \in Y$  that satisfy

$$\frac{\#\{i = 1, \dots, n : \alpha_i \geq \alpha_n^y\}}{n} > \delta \quad (3)$$

where  $\alpha_1, \dots, \alpha_{n-1}$  are the strangeness measure of the training examples and  $\alpha_n^y$  is the strangeness measure of  $z_n$  labeled  $y$ . Intuitively, we can say that  $z_n$  can take any of the labels in  $\Gamma$  at  $1 - \delta$  confidence level.

“Uncertain” *region predictor* are region predictor that has more than one label. The region predictor can also be an empty set.

*rTCM* defines a randomized region predictor  $\Gamma$  such that for any label  $y \in Y$ ,

1. Include label  $y$  in  $\Gamma$  when

$$\frac{\#\{i = 1, \dots, n : \alpha_i > \alpha_n^y\}}{n} > \delta$$

2. Do not include  $y$  in  $\Gamma$  when

$$\frac{\#\{i = 1, \dots, n : \alpha_i \geq \alpha_n^y\}}{n} \leq \delta$$

3. Otherwise, include  $y$  in  $\Gamma$  with probability

$$\frac{\#\{i = 1, \dots, n : \alpha_i \geq \alpha_n^y\} - n\delta}{\#\{i = 1, \dots, n : \alpha_i = \alpha_n^y\}}$$

$y$  is included when the above expression is greater than a random number drawn from a uniform distribution on  $[0, 1]$ .

A *rTCM* is proven to have an error probability  $\delta$  at trial  $n = 1, 2, \dots$ , and this error probability at each trial is independent of other trials [17].

Like the *TCM*, *rTCM* predicts the label of an example as the one with the largest p-value with a confidence measure equals to one minus the second largest p-value and credibility equals to the largest p-value [3]. If the region predictor is of size larger than one, we know that the prediction has low confidence.

We will show the usefulness of these additional information for stream-based active learning in Section 8.2. More details and properties of region predictor and *rTCM* can be found in [17].

### 4. Incremental Support Vector Machine

To improve the speed of SVM, many incremental algorithms have been proposed. Earlier implementations of incremental SVM make use of the properties of the support vectors and the distribution knowledge of the sample space [20] to reduce the size of the training set as more examples are added. In some cases, only the historical support vectors and the new examples are kept for training the classifier [14]. These methods, however, require re-training the classifier using the new training data-set.

Recently, implementations of incremental SVM that avoid retraining all the training examples have been suggested [4] [7] [1].

Fung and Mangasarian [4] proposed a fast and simple incremental support vector machine that modifies the current linear classifier by “both retiring old data and adding new data”. Their method uses a non-standard SVM formulation that classifies points by assigning them to the closest of two parallel hyper-planes that are pushed apart as far as possible. However, their non-standard SVM does not provide the standard strangeness measure, a set of Lagrange multipliers for the training examples, required by TCM using SVM as the base classifier.

Kivinen et. al. [7] considered online learning in a Reproducing Kernel Hilbert Space using classical stochastic gradient descent within a feature space and derive a rate of convergence and error bound. However, the error bound does not corroborated well with their experiments [7].

Cauwenberghs and Poggio [1] proposed the implementation of incremental SVM using adiabatic increment. We choose to use this incremental SVM implementation since

1. it can be easily integrated into TCM.
2. it constructs the exact solution recursively as a new example is added.

We review the main idea of adiabatic increment in the next section.

## 5. Incremental SVM by Adiabatic Increment

Given a set of labeled examples

$$\{z_1, \dots, z_n\} = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

where  $y_i \in \{-1, 1\}$  for  $i = 1, 2, \dots, n$  such that the two classes of examples are not linearly separable, an optimal hyperplane is constructed by minimizing the functional

$$W(\xi) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \quad (4)$$

where  $w$  is the hyperplane normal vector, the constant  $C > 0$  and  $W(\xi)$  subjects to the constraints

$$y_i(w \cdot \Phi(x_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, 2, \dots, n \quad (5)$$

where  $\Phi(x_i)$  maps  $x_i$  into the feature space and  $b$  is an offset. Introducing Lagrange multipliers,  $\alpha_i \geq 0, i = 1, \dots, n$ , for each of the constraints above, the dual form of the above optimization problem is expressed as

$$\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j \alpha_i \alpha_j \Phi(x_i) \cdot \Phi(x_j) - \sum_{i=1}^n \alpha_i \rightarrow \min_{\alpha_i} \quad (6)$$

$$\sum_{i=1}^n y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C, i = 1, \dots, n \quad (7)$$

Apply the Lagrange method again on the dual problem, we get

$$W(\alpha, b) = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j \alpha_i \alpha_j \Phi(x_i) \cdot \Phi(x_j) - \sum_{i=1}^n \alpha_i + b \sum_{i=1}^n y_i \alpha_i \quad (8)$$

Replacing  $y_i y_j \Phi(x_i) \cdot \Phi(x_j)$  with  $Q_{ij}$  ( $(i, j)$ -entry of the kernel matrix,  $Q$ ),

$$W(\alpha, b) = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i Q_{ij} \alpha_j - \sum_{i=1}^n \alpha_i + b \sum_{i=1}^n y_i \alpha_i \quad (9)$$

From the slack-variables,  $\xi_i$ , we get the constraints  $\alpha_i \leq C, i = 1, \dots, n$ .

$W(\alpha, b)$  is minimized given the Karush-Kuhn-Tucker (KKT) conditions

$$g_i = \frac{\partial W}{\partial \alpha_i} = \sum_{j=1}^n Q_{ij} \alpha_j + y_i b - 1 \quad (10)$$

$$\begin{cases} > 0 & \alpha_i = 0 \\ = 0 & 0 \leq \alpha_i \leq C \\ < 0 & \alpha_i = C \end{cases}$$

$$\frac{\partial W}{\partial b} = \sum_{j=1}^n y_j \alpha_j = 0 \quad (11)$$

The first KKT condition partitions the training data into three sets:

1. the set of support vectors that lies on the margin,  $S$ , when  $0 < \alpha_i < C$ .
2. the set of support vectors that lies in the margin,  $E$ , when  $\alpha_i = C$ .
3. the remaining set,  $R$ , when  $\alpha_i = 0$ .

The main idea of adiabatic increment is that all the examples in the training set do not violate the KKT conditions when a new example  $z_{n+1}$  is added to this training set when its Lagrange multiplier  $\alpha_{n+1}$ , initially set to zero, is incremented. In order to do so, some ‘‘bookkeeping’’ [1] procedure is necessary.

When the value of  $\alpha_{n+1}$  increases, the ‘‘bookkeeping’’ procedure updates the values of  $g_i$  (Equation (10)) and  $\alpha_i$  for  $i = 1, 2, \dots, n, n + 1$ . In doing so, an example may be transferred from one set to the other two sets accordingly.

More derivation details and implementation issues such as the computation of the Jacobian inverse,  $\mathcal{J}$ , by recursion, and maintaining of a smaller remaining set,  $R$ , to improve computational efficiency, are found in [1].

## 6. Online vs Incremental Learning

Before we introduce our TCM implementation in Section 7, we will like to point out some distinctions between online and incremental learning. They should not be used interchangeably.

We need to distinguish between the learning *setting* and learning *algorithm*. Two questions can be asked:

1. Is the learning *setting* (i) online or (ii) offline ?
2. Is the learning *algorithm* (i) incremental or (ii) batch ?

In an online setting, data enters the learner sequentially, as packets of data or individual datum. This is different from the offline setting when the whole training set is assumed to be available to the learner.

For the online and offline settings, the learning algorithm can be incremental. In both cases, the data are iteratively given to the learner to learn a general rule, the rule learned during the current trial is constructed using some information from the previously learned rule and the new data. This type of learning is *algorithmically incremental*.

Earlier methods [20] [14] on incremental SVM are not algorithmically incremental. They require the re-training of the classifier using all the available training examples at each trial. These are actually *batch algorithms used in an online setting*.

Later methods such as incremental proximal support vector classifier [4], Kivenen and et. al.’s online learning [7] and the incremental SVM by adiabatic increments [1] are all algorithmically incremental. We note that since SVM is a convex optimization problem, a correctly formulated incremental algorithm should always achieve the optimal solution.

## 7. Incremental Online Transductive Confidence Machine

As we pointed out in Section 1 that the practical distinction between transduction and induction is whether we extract and store the general rule or not. More specifically, there is critical information required by the transduction methods that is also computed in the induction methods.

Recent developments in incremental (inductive) classifier, such as the adiabatic incremental SVM, give us some hints on how to go about improving the computational efficiency of transduction. Instead of extracting and storing the general rule from the training set and repeating this process in the online setting, we could compute and store critical information efficiently from the current training set to be used for any transduction process later.

Let  $T$  be the current training set with  $n - 1$  labeled examples,  $\{\alpha, b\}$  be the current solution where  $\alpha$  is the set of Lagrange multipliers and  $b$  is the offset,  $\mathcal{J}$  be the current Jacobian inverse,  $Q$  be the kernel matrix,  $S$  and  $E$  be the current two support vector sets (on and within the margin, respectively) and  $R$  be the current remaining vector set.

By integrating the procedure of the adiabatic incremental learning into the algorithm for TCM (Section 2), we get a more “tightly-coupled” TCM algorithm:

**Algorithm (One transduction process for a new unlabeled example):**

Input:  $z_n = (x_n, ?)$ ,  $T$ ,  $\{\alpha, b\}$ ,  $\mathcal{J}$ ,  $Q$ ,  $S$ ,  $E$ ,  $R$ .

1. Keep a copy of  $\{\alpha, b\}$ ,  $\mathcal{J}$ ,  $Q$ ,  $S$ ,  $E$ ,  $R$  for re-initialization at the end of each FOR loop
2. FOR  $y = 1$  to  $c$  (number of possible classes)
  - (a) Label  $z_n$  as  $y$  (i.e.  $z_n = (x_n, y)$ )

- (b) Initialize  $\alpha_{z_n}^y$  to zero.
- (c) Compute  $Q_{z_n, j}$  for all  $j \in S$  (the set of support vectors) and include it into  $Q$ .
- (d) Compute  $g_{z_n}$  using Equation (10).
- (e) If  $g_{z_n} > 0$ ,
  - i.  $\alpha_{z_n}^y = 0$ , and goto (g)
- (f) Elseif  $g_{z_n} \leq 0$ , apply the largest possible increment  $\Delta\alpha_{z_n}^y$  so that (the first) one of the following conditions occurs:
  - i.  $g_{z_n} = 0$ : Add  $z_n$  to  $S$ , update  $R$  accordingly, goto (g).
  - ii.  $\alpha_{z_n}^y = C$ : Add  $z_n$  to  $E$  and goto (g).
  - iii. Do “bookkeeping” among  $S$ ,  $E$  and  $R$ . Update  $\mathcal{J}$  if  $S$  changes
  - iv. Repeat (f) until no more changes to  $S$ ,  $E$  and  $R$ .
- (g) Compute p-values for  $z_n = (x_n, ?)$  labeled  $y$  using  $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_{n-1}, \alpha_{z_n}^y\}$ .
- (h) Test whether  $y$  is in the region predictor of  $z_n = (x_n, ?)$  with confidence level  $1 - \delta$  using  $rTCM$ .
- (i) Store the p-value of  $z_n$  labeled  $y$ .
- (j) Re-initialize  $\{\alpha, b\}$ ,  $\mathcal{J}$ ,  $Q$ ,  $S$ ,  $E$ ,  $R$  to their respective input values.

ENDFOR

3. Assign the label with the largest p-value to example  $z_n$ .
4. Calculate the confidence as  $1 -$  second largest p-value.

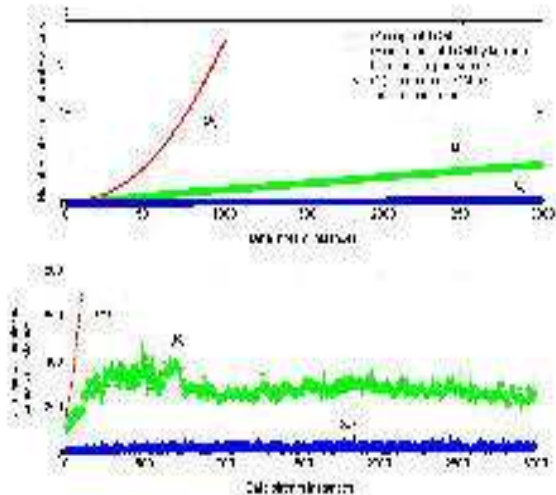
In the next section, we use the acronym “TCM” to mean the *online randomized TCM using SVM as the base classifier*.

## 8. Experiments

This section describes experimental data concerning the comparative efficiency for our novel adiabatic incremental TCM method using SVM, and its utility for stream-based active learning.

### 8.1. Comparison of computational cost and classification performance

We generated a binary data-set using Musicant’s NDC (normally distributed clustered) data-set generator [9]. The data is generated as clusters of normally distributed points in  $\mathbb{R}^n$  with an adjustable linear separability. The values of each dimension are scaled to range in  $[-1, 1]$ . The data-set consists of 3000 points in a 32 dimensional input space and the experiments are performed in MATLAB 6.5.



**Figure 1. Comparison of the (upper graph) total number of updating iterations and (lower graph) number of updating iterations at each data stream instance required for (A) original TCM, (B) incremental TCM by keeping historical support vectors and (C) incremental TCM by adiabatic increment. (Results for (A) only show 1000 instances in the upper graph and 100 instances in the lower graph as the high rate of increment affects the visualization of results for (B) and (C))**

We compare the computational cost and performance of (A) the original TCM, (B) the incremental TCM by keeping the historical support vectors, and (C) the incremental TCM by adiabatic increment described in Section 7. As we mentioned in Section 6, incremental SVM can be used in an off-line setting. We use the adiabatic incremental SVM, that has comparable computational cost as other quadratic programming implementations of SVM in an off-line setting, as the base classifier in (A) and (B). For all experiments,  $\delta$  is set to 0.1 i.e. 90% confidence level. For SVM, we use a Gaussian kernel and  $C$  is set to 10.

The computational cost is based on the number of iterations of step 2(f), the adiabatic step, which is the most expensive step in the three implementations. The upper graph in Figure (1) shows the total computational cost of the three implementations at each instance when a new example is included into the training set of the learner. We see that our implementation (C) requires much less number of total iterations and the rate of increment has a much gentler slope than implementation B while the the total iterations of original TCM increases exponentially. From the lower graph in Figure (1), we see that at each streaming data instance,

the number of iterations remains small and stable for implementation (C).

We note here that each iteration does not corresponds to a fix time unit since at each iteration some matrix computations are required such that the computation cost depends mainly on the size of the inverse Jacobian,  $\mathcal{J}$ , that depends on the number of support vectors, and the number of training examples. Maintaining a smaller set  $R$  and assuming a “concept drift” setting, likely to occur in real world problems, are some ways to provide an upper bound for the computational cost at each iteration.

We observe from Figure (2) that using adiabatic increment for TCM and original TCM have very similar learning curves for accuracy, error and uncertainty predictions. The slight differences are due to the randomized nature of our TCM implementation.

We also note from Figure (2) that the number of uncertain predictions are much higher than the accurate predictions for implementation (B) while its error predictions are higher than the other two implementations. This is the result of information lost when data instances that are not support vectors previously are discarded, which may become useful later.

Theoretically, the data samples are no longer i.i.d. and the error bound for TCM is no longer valid. Empirically, we observed that despite the fact that the training examples used at each trial of (B) are smaller than both (A) and (C) in a long data stream, its classification performance is dubious.

## 8.2. Application of TCM: Stream-based active learning

TCM has been used to perform active learning in both off-line batch setting [5] and online streamed-based setting [6]. For stream-based active learning, unlabeled examples are provided to the learner one by one, and the learner must decide whether or not to request its label and add it to the training set.

The goals for active learning are two fold : (i) less computation due to smaller training sets without penalizing the performance of the classifier, and (ii) reducing the human labeling efforts and cost.

We recall that in Section 3, we describe a region predictor as a function, when given an unlabeled test input and a confidence level, returns a set of possible predictions with a degree of confidence and uncertain region predictor are region predictor that has more than one label. Our stream-based active learning method takes into account the *deviation from uncertain region predictor* of each unlabeled example provided to the learner. This deviation measure makes use of the p-values computed for all possible labels for an unlabeled example by TCM.

Given a binary classification problem,  $\mathcal{U}(e) = |p_i - p_j|$ ,

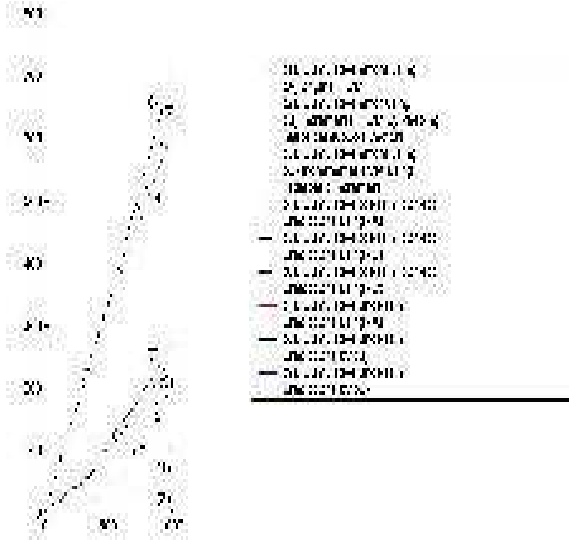


Figure 2. Comparison of the training performance of (A) original TCM, (B) incremental TCM by keeping the historical support vectors and (C) incremental TCM using adiabatic increment. (A) and (C) have very similar learning curves. Slight difference is due to the randomized nature of the algorithm.

where  $p_i$  is the p-value for an unlabeled example  $e$  labeled as  $i$  and  $p_j$  is the p-value for the example labeled as  $j$ , is a measure of the deviation from an uncertain region predictor. As  $\mathcal{U}(e)$  increases from 0 to 1, the example  $e$  becomes more likely to be a particular label.

**Selection Criteria:** For a given example  $e$ , if  $\mathcal{U}(e) \leq \eta$  (a threshold value), add  $e$  to the training set,  $T$ .

The theoretical justification and empirical studies of using TCM on stream-based active learning is given in [6].

Here, we use the binary classification problem, Image, from the benchmark collection in [12], to demonstrate stream-based active learning using TCM. The Image data-set consists of 2310 examples in a 18 dimensional input space. In the benchmark collection, there are 20 different training/testing partitions of Image data-set such that each training set contains 1300 examples and each testing set contains 1010 examples.

In our experiment, the learner is first provided with one randomly chosen example from each class. Subsequently, the learner is provided with a stream of unlabeled data. At each trial, a new unlabeled point from the stream is introduced to the learner. The data stream is of length 1300.

In Table 1, we compare the number of examples selected by the active learner and their prediction accuracy at the

Method	No of selected examples	Accuracy Estimate
Standard	1300	95.65±0.56
Random	652.00±14.98	94.04±0.88
Active	273.30±49.27	94.33±1.05

Table 1. Comparison of (i) using all samples (Standard), (ii) random sampling (Random) and (iii) active learning (Active). Accuracy and number of selected examples are compared. Estimates are averaged over 20 trials.

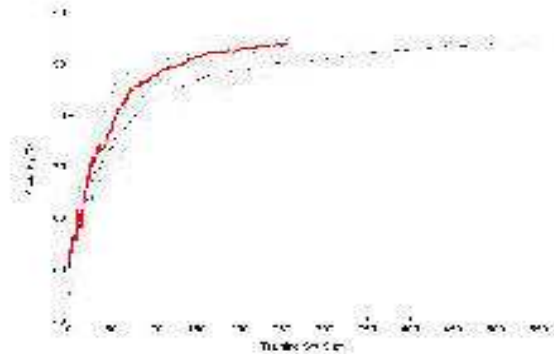


Figure 3. Our active learning (upper solid curve with error bars (dotted curve)) shows very competitive performance compared to the random sampling (lower solid curve) when the same number of examples are selected.

end of the data stream. About one-fifth of the total data is selected using active learning and the performance is almost as good as using all the examples in the data stream. The total number of examples selected is much less than random sampling and the accuracy estimate of active learning is comparable.

Figure (3) shows that active learning using TCM has very competitive performance compared to random sampling when the same number of examples are selected.

## 9. Conclusion

The use of transductive learning in temporal and real-time data mining is hindered by the fact that existing transduction methods are expensive and inefficient computation compared to inductive learning techniques. One objective of this paper is to encourage temporal data-mining researchers to explore the use of transduction methods in their

works. In order to do so, we introduce a fast transduction method to overcome the inefficiency problem of transduction.

Our transduction method is a fast incremental TCM based on adiabatic incremental SVM. Our method is empirically shown to be computationally efficient and its performance is consistent with the standard TCM implementation. Besides being a classifier, TCM provides additional useful statistical information about the data that it has processed. We demonstrate the usefulness and feasibility of such statistical information for stream-based active learning. This information can be useful to other temporal and real-time data mining.

We also show that incremental SVM, by keeping historical support vectors, when integrated into TCM do not have consistent performance compared to the standard TCM implementation. Hence, the integration of incremental algorithm into transduction methods must be cautiously done to achieve both a reduction in the computational cost and a competitive performance.

Transduction can then become a viable technique for learning in data stream.

## Acknowledgement

The first author would like to thank Dr Alex Gammerman and Dr. Volodya Vovk for the manuscript of their book "Algorithmic learning in a random world".

## References

- [1] Cauwenberghs, G. and Poggio, T. Incremental support vector machine learning, *Advances in Neural Information Processing Systems 13*, MIT Press, 409-415, 2000.
- [2] Gammerman A., Vovk. V. and Vapnik. V. Learning by Transduction. *Uncertainty in Artificial Intelligence*, Proc. of the Fourteenth Conference, 148-155. 1998.
- [3] Gammerman A. and Vovk V. Prediction algorithms and confidence measures based on algorithmic randomness theory, *Theoretical Computer Science* 287, 209-217, 2002.
- [4] Fung G. and Mangasarian O.L. Incremental support vector machine classification. Proc. of the second SIAM ICDM, 247-260, 2002.
- [5] Ho S.-S. and Wechsler H. Transductive Confidence Machine for active learning, Proc. of Int. Joint Conf. on Neural Networks, 2, 1435-1440, 2003.
- [6] Ho S.-S. and Wechsler H. Stream-based active learning using algorithmic randomness theory. (Manuscript in preparation).
- [7] Kivinen J., Smola A. and Williamson R. Online learning with kernels, *Advances in Neural Information Processing Systems 14*, MIT Press, 785-792, 2001
- [8] Li, M., Vitanyi, P. An Introduction to Kolmogorov Complexity and Its Applications, 2nd Edition. Springer-Verlag, 1997.
- [9] Musicant, D. R. NDC: Normally Distributed Clustered Datasets, Computer Sciences Department, University of Wisconsin, Madison, <http://www.cs.wisc.edu/dmi/svm/ndc/>, 1998.
- [10] Papadopolous H., Vovk V. and Gammerman A. Qualified predictions for large data sets in the case of pattern recognition. Proc. of the Int. Conf. on Machine Learning and Applications , 159-163, 2002.
- [11] Proedrou, K., Nouretdinov, I., Vovk, V., Gammerman, A. Transductive confidence machine for pattern recognition. Proc. 13th European Conference on Machine Learning. Vol. 2430. pp. 381-390, 2002
- [12] Raetsch G., Onoda T. and Mueller K.R. Soft margins for Adaboost, *Machine Learning* 42, 3, 287-320, 2001.
- [13] Saunders C., Gammerman A. and Vovk V. Computational efficient transductive machines, *Algorithmic Learning Theory*, 11th International Conference, Lecture Notes in Computer Science, 1968, Springer, 325-333, 2000.
- [14] Syed N., Liu H. and Sung K.-K. Incremental learning with support vector machines, Proc. Workshop on support vector machines at IJCAI-99, Stockholm, Sweden, 1999.
- [15] Vapnik, V. *Statistical Learning Theory*, Wiley-Series, 1998.
- [16] Vovk V., Gammerman A. and Saunders C. Machine-learning applications of algorithmic randomness. Proc. of the Sixteenth Int. Conf. on Machine Learning, 444-453, 1999.
- [17] Vovk V. On-line confidence machines are well-calibrated. Proc. 43th IEEE Symposium on Foundations of Computer Science, 187-196, 2002.
- [18] Vovk V, Gammerman A. and Shafer G. Algorithmic learning in a random world (Manuscript), July 2004.
- [19] Weerahandi, S. Exact statistical methods for data analysis. Springer-Verlag, 1994.
- [20] Xiao, R., Wang, J. and Zhang, F. An approach to incremental svm learning algorithm, 12th IEEE Int. Conf. on Tools with Artificial Intelligence, 2000.

# Learning Linear Dependency Trees from Multivariate Time-series Data

Jarkko Tikka and Jaakko Hollmén  
Helsinki University of Technology  
Laboratory of Computer and Information Science  
P.O. Box 5400, FIN-02015 HUT, Finland  
tikka@mail.cis.hut.fi, Jaakko.Hollmen@hut.fi

## Abstract

Representing interactions between variables in large data sets in an understandable way is usually important and hard task. This article presents a methodology how a linear dependency structure between variables can be constructed from multivariate data. The dependencies between the variables are specified by multiple linear regression models. A sparse regression algorithm and bootstrap based resampling are used in the estimation of models and in construction of a belief graph. The belief graph highlights the most important mutual dependencies between the variables. Thresholding and graph operations may be applied to the belief graph to obtain a final dependency structure, which is a tree or a forest. In the experimental section results of the proposed method using real-world data set were realistic and convincing.

## 1. Introduction

Large data sets are available from many different sources, for example from industrial processes, economy, mobile communications network, and environment. Deeper understanding of the underlying process can be achieved by exploring or analyzing the data. Economical or ecological benefits are a great motivation for the data analysis.

In this study, dependencies between the variables in data set are analyzed. The purpose is to estimate multiple linear regression models and learn a linear dependency tree or forest of the variables. The dependency structure clearly shows how a change in a value of one variable induces changes in values of other variables. This might be useful information in many cases, for instance, if values of some variable cannot be controlled directly.

The multiple linear regression models have a couple of advantages. The dependencies in linear models are easy to interpret. In addition, processes may be inherently linear or

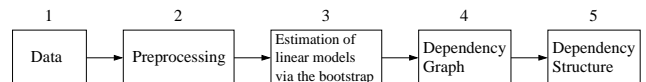


Figure 1. The flow chart of the proposed method.

over short ranges many processes can be approximated by a linear model.

A flow chart of the methodology proposed in this study is presented in Figure 1. The method consists of five phases. First, there should be some multivariate data available. The data do not necessarily have to be time-series data, although time-series data is used as an example in this study.

The second phase deals with the preprocessing of data. Some operations have to be usually performed on measurements before they can be analyzed mathematically. Some measurements may be missing or measurements can be noisy.

In the third phase, as many multiple linear regression models as there are variables in the data are estimated. Each variable is a dependent variable in turn and the rest of the variables are possible independent variables. The most significant independent variables for each model are selected using the bootstrap and a sparse regression algorithm. The relative weights of the regression coefficients are computed from the bootstrap replications. The relative weight of the regression coefficient measures a belief that the corresponding independent variable belongs to the estimated linear model.

In the fourth phase, a belief graph is constructed from the relative weights of the regression coefficients. The belief graph represents the strength of the dependencies between the variables. In the belief graph there are as many nodes as there are variables in the data. The relative weights define arcs of the belief graph. A predefined threshold value and a moralizing operation are applied to the belief graph

resulting a moral graph or a final dependency graph.

Finally, a dependency structure of the variables is calculated from the dependency graph. A set of variables, which forms a multiple linear regression model, belongs to a same maximal clique. However, the formulation of final dependency structure is restricted such that the dependencies cannot form circles in a final structure i.e. the variable cannot be dependent on itself through the other variables. Thus, the final dependency structure is a tree or a forest.

The rest of the article is organized as follows. In Section 2 a few other similar studies are briefly described. The multiple linear regression model and sparse regression algorithms are introduced in the beginning of Section 3, followed by the bootstrap and the computation of the relative weights of the regression coefficients. The construction of linear dependency tree or forest is proposed in Section 4. The proposed method is applied to real-world data set. The description of data and the results of experiments are shown in Section 5. The experiments mainly serve an illustrative example of the proposed methodology. Conclusion and final remarks are in Section 6.

## 2 Related work

To our knowledge, novelty of this work is in the sparse construction of linear models and the application of the bootstrap. Several studies about dependencies between the variables in multivariate data are accomplished, for example [3], [13], [11], [16], and [20].

Dependency trees are also used in [3]. A method which approximates optimally a  $d$ -dimensional probability distribution of the  $d$  variables is shown. Each variable can only be dependent on at most one variable in that model, when in this study one variable can be dependent on several variables.

Belief networks are discussed in [13]. The belief network induces a conditional probability distribution over its variables. The belief networks are directed and acyclic. Dependency networks which can be cyclic are presented in [11]. In both belief and dependency networks the variables are conditioned upon its parent variables. The directed dependency means that changes in the parent has effect on the child. The undirected dependency means that changes are induced into the both directions. In this study, continuous variables are only modeled, whereas the belief and the dependency network can be used with discrete variables.

Independent variable group analysis (IVGA) is proposed in [16]. In that approach the variables are clustered. The variables in one cluster are dependent on each other but they are independent on the variables which belong to other clusters. In IVGA, the dependencies between the groups or clusters are ignored and the dependencies in each group can be modeled in different ways.

Structural equation modeling (SEM) [20] is another technique to investigate relationships between the variables. SEM provides a methodology to test a plausibility of hypothesized models. The predefined dependencies between the variables are investigated using the SEM, when the dependencies are learned from the data using the method proposed in this study. Structural Equation models can consist of both observed and latent variables. The latent variables can be extracted from the observed ones using for example the factor analysis. Observed variables are only modeled in this study.

## 3 Methods

### 3.1 Multiple linear regression

The dependencies between the variables are modeled using the multiple linear regression. The model is

$$y_t = \beta_1 x_{t,1} + \beta_2 x_{t,2} + \dots + \beta_k x_{t,k} + \epsilon_t, \quad (1)$$

where  $y_t$  is the dependent variable,  $x_{t,i}, i = 1, \dots, k$  are the independent variables,  $\beta_i, i = 1, \dots, k$  are the corresponding regression coefficients, and  $\epsilon_t$  is normally distributed random noise with zero mean and unknown variance  $\epsilon_t \sim N(0, \sigma^2)$ . The index  $t = 1, \dots, N$  represents the  $t$ th observation of the variables  $y$  and  $x_i$  and  $N$  is the sample size.

Equation (1) can also be written in matrix form as follows

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}. \quad (2)$$

Here we assume that the variables are normalized to zero mean and thus, there is no need for a constant term in models (1) and (2).

The ordinary least squares (OLS) solution is

$$\mathbf{b}_{OLS} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad (3)$$

where  $\mathbf{b}_{OLS} = [b_1, \dots, b_k]$  is the best linear unbiased estimate of the regression coefficients.

### 3.2 Linear sparse regression

The usual situation is that the available data are  $(\mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{y})$  and the linear regression model should be estimated. The OLS estimates are calculated using all the independent variables. However, the OLS estimates may not always be satisfactory. The number of possible independent variables may be large and there are likely non-informative variables among them.

The OLS estimates have a low bias but a large variance. The large variance impairs the prediction accuracy. The prediction accuracy can sometimes be improved by shrinking

some regression coefficients toward zero, although at the same time the bias increases [4]. The models with too many independent variables are also difficult to interpret. Now, the objective is to find a smaller subset of independent variables that have the strongest effect in the regression model.

In the subset selection regression only a subset of the independent variables are included to the model, but it is an inefficient approach if the number of independent variables is large. The subset selection is not robust because small changes in the data can result in very different models. More stable result can be achieved using the nonnegative garrote [2]. The garrote also eliminates some variables and shrinks other coefficients by some positive values.

Ridge regression [12] and lasso [22] algorithms produce a sparse solution or at least shrink estimates of the regression coefficients toward zero. Both algorithms minimize a penalized residual sum of squares

$$\operatorname{argmin}_{\beta} \left\{ \|\mathbf{y} - \mathbf{X}\beta\|^2 + \lambda \sum_{i=1}^k |\beta_i|^\gamma \right\}, \quad (4)$$

where  $\gamma = 2$  in ridge regression and  $\gamma = 1$  in lasso. The tuning parameter  $\lambda$  controls the amount of shrinkage that is applied to the coefficients. The problem in Equation (4) can be represented equivalently as a constrained optimization problem. In that approach the residual sum of squares  $\|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2$  is minimized subject to

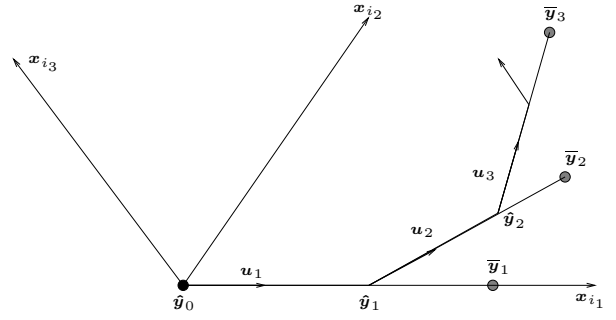
$$\sum_{i=1}^k |\beta_i|^\gamma \leq \tau, \quad (5)$$

where  $\gamma$  is the same as in Equation (4) and the constant  $\tau$  controls the amount of the shrinkage. The parameters  $\lambda$  in Equation (4) and  $\tau$  in Equation (5) are related to each other by a one-to-one mapping [10]. A large value of  $\lambda$  corresponds to a small value of  $\tau$ .

The ridge regression solution is easy to calculate, because the penalty term is continuously differentiable. The solution is

$$\mathbf{b}_{RR} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}, \quad (6)$$

where  $\mathbf{I}$  is an identity matrix, but it does not necessarily set any coefficients exactly to zero. Thus, the solution is still hard to interpret if the number of independent variables  $k$  is large. The lasso algorithm sets some coefficients to zero with a proper  $\tau$ , but finding the lasso solution is more complicated due to absolute values in the penalty term. A quadratic programming algorithm has to be used to compute the solution. Also, the value of  $\tau$  or  $\lambda$  which controls the shrinkage is strongly dependent on data. Therefore, seeking such a value may be difficult in many cases. The data-based techniques for estimation of the tuning parameter  $\tau$  are presented in [22].



**Figure 2. The progress of the LARS algorithm. The figure is reproduced from the original LARS article by Efron et al. [7].**

The lasso algorithm is not applicable if the number of possible independent variables is large. Forward stagewise linear regression (FSLR) can be used instead of lasso in that case [10]. FSLR approximates the effect of the lasso penalty  $\gamma = 1$  in Equation (4). New independent variables are added sequentially to the model in FSLR. Two constants  $\delta$  and  $M$  have to be set before iterations. The regression coefficient that diminish most the current residual sum of squares, is adjusted by amount of  $\delta$  at each successive iteration. The value of  $\delta$  should be small and  $M$  should be a relatively large number of iterations.

All the estimates of coefficients  $b_i, i = 1, \dots, k$  are set to zero in the beginning. Many of the estimates  $b_i$  are possibly still zero after  $M$  iterations. It means that corresponding independent variables are not yet added to the regression model. The solution after the  $M$  iterations is almost similar than the lasso solution with some  $\lambda$ . They are even identical in some cases [10].

The preceding methods such as ridge regression, lasso, and FSLR are introduced as the historical precursors of the Least Angle Regression (LARS) model selection algorithm. We are mainly interested in the methods producing sparse models. Thus, lasso and FSLR could be applied, but they have deficiencies compared to the LARS algorithm. The parameters  $\lambda$  or  $\tau$  in lasso and  $\delta$  and  $M$  in FSLR have to be predefined, whereas LARS is completely parameter free and it is also computationally more efficient than lasso or FSLR. However, all the three methods produce nearly same solutions. Only LARS algorithm is applied to selection of the most significant independent variables in this study.

In Figure 2 the progress of the LARS algorithm is visualized. All the variables are scaled to have zero mean and unit variance. One independent variable is added to the model in each step. First, all regression coefficients are set to zero. Then, the most correlated independent variable  $x_{i_1}$  with  $\mathbf{y}$  is found. The largest possible step in the direction

of  $\mathbf{u}_1$  is taken until some other variable  $\mathbf{x}_{i_2}$  is as correlated with the current residuals as  $\mathbf{x}_{i_1}$ . That is the point  $\hat{\mathbf{y}}_1$ . At this point LARS differs from traditional Forward Selection, which would proceed to the point  $\bar{\mathbf{y}}_1$ , but the next step in LARS is taken in a direction  $\mathbf{u}_2$  equiangular between  $\mathbf{x}_{i_1}$  and  $\mathbf{x}_{i_2}$ . LARS proceeds in this direction until a third variable  $\mathbf{x}_{i_3}$  is as correlated with the current residuals as  $\mathbf{x}_{i_1}$  and  $\mathbf{x}_{i_2}$ . Next step is taken in a direction  $\mathbf{u}_3$  equiangular between  $\mathbf{x}_{i_1}$ ,  $\mathbf{x}_{i_2}$ , and  $\mathbf{x}_{i_3}$  until a fourth variable can be added to the model. This procedure is continued as long as there are still independent variables left. So,  $k$  steps are needed for the full set of solutions i.e. the result is  $k$  different multiple linear regression models. In Figure 2  $\bar{\mathbf{y}}_i$  represent the corresponding OLS estimates from  $k$ th step. LARS estimates  $\hat{\mathbf{y}}_k$  approach but never reach OLS estimates  $\bar{\mathbf{y}}_k$ , except at the last step the LARS and OLS estimates are equivalent. The mathematical details of LARS algorithm are presented in [7].

The problem is to find the best solution from all the  $k$  possibilities which LARS returns i.e. a proper number of independent variables. This selection can be done according to the minimum description length (MDL) information criterion [9]. The variance  $\sigma^2$  of  $\epsilon_t$  is assumed to be unknown, thus, the MDL criterion is written in context of the linear regression, as presented in [9],

$$MDL(k) = \frac{N}{2} \log \|\mathbf{y} - \hat{\mathbf{y}}\|^2 + \frac{k}{2} \log N. \quad (7)$$

$\mathbf{y}$  is the dependent variable,  $\hat{\mathbf{y}}$  is the estimate of the dependent variable,  $N$  is the sample size and  $k$  is the number of added independent variables. The value of Equation (7) is calculated for all the solutions. The selected regression model minimizes Equation (7).

The Mallows  $C_p$  criterion [18], [19] is a common criterion in subset selection. However,  $C_p$  is not used, because it can select submodels of too high dimensionality [1]. A review of several other information criteria can be found from [21].

### 3.3 Bootstrap

The bootstrap is a statistical resampling method and it was introduced by Efron in [6]. The idea of bootstrap is to use sample data to estimate some statistics of the data. No assumptions are made about the forms of probability distributions in the bootstrap procedure. The statistic of interest and its distribution are computed by resampling the original data with replacement.

Bootstrapping a regression model can be done in two different ways. The methods are bootstrapping residuals and bootstrapping pairs [8]. The independent variables ( $\mathbf{x}_1, \dots, \mathbf{x}_k$ ) are treated as fixed quantities in the bootstrapping residuals approach. That assumption is strong and it

can fail even if Equation (1) for the regression model is correct. In the bootstrapping pairs approach weaker assumptions about validity of Equation (1) are made.

In the bootstrapping pairs,  $\hat{F}$  is assumed to be an empirical distribution of the observed data vectors  $(x_{t,1}, \dots, x_{t,k}, y_t)$ , where  $t = 1, \dots, N$ .  $\hat{F}$  puts probability mass of  $1/N$  on each vector  $(x_{t,1}, \dots, x_{t,k}, y_t)$ . A bootstrap sample is now a random sample of size  $N$  drawn with replacement from the population of  $N$  vectors  $(x_{t,1}, \dots, x_{t,k}, y_t)$ .

$B$  independent bootstrap samples  $(\mathbf{X}^{*i}, \mathbf{y}^{*i}), i = 1, \dots, B$  of the size  $N$  are drawn from the distribution  $\hat{F}$ . The bootstrap replications  $\mathbf{b}^{*i}$  of the estimates  $\mathbf{b}$  are computed using the LARS algorithm and the MDL information criterion. The statistic of interest or some other features of the parameters  $\mathbf{b}$  can be calculated from these  $B$  bootstrap replications.

### 3.4 Computation of relative weights of regression model

In this study, relative weights of the coefficients of multiple linear regression model are computed. The relative weights are calculated from the bootstrap replications as follows

$$\mathbf{w} = \frac{1}{B} \sum_{i=1}^B \frac{|\mathbf{b}^{*i}|}{\mathbf{1}^T |\mathbf{b}^{*i}|}. \quad (8)$$

$B$  is the number of bootstrap replications and  $\mathbf{b}^{*i}$  is the  $i$ th bootstrap replication of coefficients  $\mathbf{b}$ . The absolute values are taken over all the components of vector  $\mathbf{b}^{*i}$ . There is a sum of the absolute values of coefficients in the denominator.  $\mathbf{1}$  is a vector of ones and the length of the vector is the same as the length of the vector  $\mathbf{b}^{*i}$ . All the components of vector  $|\mathbf{b}^{*i}|$  are divided by the previous sum. These operations are done for every bootstrap replication and the scaled bootstrap replications are added together. This sum is divided by the number of bootstrap samples  $B$ . The result is a vector  $\mathbf{w}$ , which includes the relative weights of the coefficients  $\mathbf{b}$ .

There is a relative weight  $w_i$  for the each possible independent variable  $\mathbf{x}_i, i = 1, \dots, k$  in the vector  $\mathbf{w}$ . The value of each  $w_i$  is within the range  $w_i \in [0, 1]$  and  $\sum_i w_i = 1$ . The relative weight of the independent variable is a measure of the belief that the independent variable belongs to the estimated linear model. The independent variable can be rejected from the estimated model if the value of  $w_i$  is zero or under a predefined threshold value. The most significant independent variables have the largest relative weights. In this study the variables are scaled to have unit variance, therefore, the regression coefficients are comparable to each other and their absolute values can be used as a measure of significance.

The vector of relative weights can also be regarded as a discrete probability distribution. From the probability distribution it can be seen which independent variables are likely to be included to the final linear sparse regression model.

## 4 Learning a linear dependency structure

### 4.1 Constructing a belief graph

Let us assume now that there are data  $D$  available, which have  $k+1$  variables and  $N$  measurements for each variable. The objective is to find multiple linear regression models among the variables. Each variable is the dependent variable in turn and the rest of the variables are the possible independent variables. So, the following models have to be estimated.

$$\begin{aligned}\hat{\mathbf{x}}_1 &= b_2^1 \mathbf{x}_2 + b_3^1 \mathbf{x}_3 + \dots + b_k^1 \mathbf{x}_k + b_{k+1}^1 \mathbf{x}_{k+1} \\ \hat{\mathbf{x}}_2 &= b_1^2 \mathbf{x}_1 + b_3^2 \mathbf{x}_3 + \dots + b_k^2 \mathbf{x}_k + b_{k+1}^2 \mathbf{x}_{k+1} \\ &\vdots \\ \hat{\mathbf{x}}_j &= b_1^j \mathbf{x}_1 + \dots + b_{j-1}^j \mathbf{x}_{j-1} + b_{j+1}^j \mathbf{x}_{j+1} + \dots \\ &\quad b_{k+1}^j \mathbf{x}_{k+1} \\ &\vdots \\ \hat{\mathbf{x}}_k &= b_1^k \mathbf{x}_1 + b_2^k \mathbf{x}_2 + \dots + b_{k-1}^k \mathbf{x}_{k-1} + b_{k+1}^k \mathbf{x}_{k+1} \\ \hat{\mathbf{x}}_{k+1} &= b_1^{k+1} \mathbf{x}_1 + b_2^{k+1} \mathbf{x}_2 + \dots + b_{k-1}^{k+1} \mathbf{x}_{k-1} + b_k^{k+1} \mathbf{x}_k\end{aligned}$$

The relative weights of the regression coefficients are computed for all the above  $k+1$  linear models as it is described in Sections 3.2-3.4. A belief graph  $G_b$  is constructed from these  $k+1$  vectors of the relative weights.

Each variable of data  $D$  is presented as a node in the belief graph  $G_b$ . The weighted arcs between the nodes are obtained from the nonzero relative weights. Thus, the weights of arcs measure the strength of the belief that there exists a linear dependency between the corresponding two variables. The directions of dependencies are from the independent variable to the dependent variable.

The dependencies or the number of arcs in the belief graph can be reduced by setting some threshold value  $\lambda$  for the relative weights. The relative weight is set to zero if it is below the threshold and the rest of the relative weights are set to unity. This means that remaining dependencies are treated as equally important thereafter. The belief graph  $G_b$  becomes unweighted directed graph  $G_d$  after using the threshold  $\lambda$ . However, some dependencies may be bidirectional. The value of threshold is not estimated according to some defined principle. A suitable value for  $\lambda$  is decided by exploring the values of relative weights. The purpose is to find such value for  $\lambda$  that minor changes in  $\lambda$  would not cause major changes in the graph  $G_d$ .

The direct use of the full information in the belief graph will be studied further.

### 4.2 Constructing a moral graph

The following idea of constructing an undirected and a moral graph from the belief graph is adapted from [13]. Let  $V_i, i = 1, \dots, k$  stand for a node or a variable in the graphs. The directions of dependencies can be discarded from  $G_d$  and the result is an unweighted undirected graph  $G_u$ . It can be assumed now that two variables  $V_i$  and  $V_j$  belong potentially to the same linear model if they are connected by an arc in  $G_u$ . We mean that the variables belong possibly to the same set of variables which forms one linear sparse regression model. That is, the roles of the variables  $V_i$  and  $V_j$  either as independent variable or dependent variable are not specified yet.

Let us assume that a variable  $\mathbf{x}_{j_3}$  is the actual dependent variable. A possible regression model is  $\mathbf{x}_{j_3} = \beta_{j_1} \mathbf{x}_{j_1} + \beta_{j_2} \mathbf{x}_{j_2} + \phi$ , where  $\phi$  is a function of the other independent variables and noise. When variables  $\mathbf{x}_{j_1}$  and  $\mathbf{x}_{j_2}$  are considered as the dependent variables, it is possible that the dependency with the variable  $\mathbf{x}_{j_3}$  is found, but the dependencies between  $\mathbf{x}_{j_1}$  and  $\mathbf{x}_{j_2}$  are ignored in both cases. However, all three variables  $\mathbf{x}_{j_1}$ ,  $\mathbf{x}_{j_2}$ , and  $\mathbf{x}_{j_3}$  belong potentially to the same linear model. An arc can be added to connect the corresponding nodes in  $G_u$ . The added arc is called a moral arc. A moral graph  $G_m$  is obtained when all moral arcs have been added to  $G_u$ . The moral arcs are added to  $G_u$  according to the following procedure.

- Create a directed graph  $G'_u$  from  $G_u$ . The directions of dependencies are set to graph  $G'_u$  such that there do not exist cycles. For each node  $V_i$ , find its parents  $\mathcal{P}_{V_i}$  in  $G'_u$ . Connect each pair of nodes in  $\mathcal{P}_{V_i}$  by adding undirected arcs between the corresponding nodes in  $G_u$ .

In this study, the graph  $G'_u$  is created from  $G_u$  such that the parent  $V_i$  has a smaller index than the child  $V_j$  i.e.  $i < j$  in  $G'_u$ . This restriction confirms that the relationships can be interpreted correctly and the number of added moral arcs is reasonable. The parent and child relationships can be defined differently to  $G'_u$  as above and it likely results in a dissimilar moral graph and a final dependency structure. The final dependency structure can be constructed as well from  $G_u$  as from  $G_m$ . Basically, sparser models are obtained from  $G_u$ , but the moral arc addition can give additional useful information in some cases.

### 4.3 Constructing final linear models

The objective is to find multiple linear regression models among the variables in the data  $D$ . The linear models or

the sets of variables are sought from the unweighted undirected graph  $G_u$  or from the moral graph  $G_m$ . The variables, which are interpreted to belong to the same model, are parts of the same maximal clique. A subgraph of  $G_u$  or  $G_m$  is called a clique if the subgraph is complete and maximal. A subgraph is complete, if every pair of nodes in the subgraph is connected by an arc. The clique is maximal, if it is not a subgraph of the larger complete subgraph [13].

An algorithm, which can be used to generate all maximal cliques from an arbitrary undirected graph, is presented in detail in [15]. A short description of the algorithm is given in the next two paragraphs.

Let  $C_n$  stand for a list of all cliques which include  $n$  nodes. The algorithm starts by forming all 2-cliques. All pairs of nodes which are connected by an arc are 2-cliques. There exists 3-clique if two 2-cliques have one node in common and two sole nodes are connected. For example, if there are cliques  $\{V_1, V_2\}$ ,  $\{V_1, V_3\}$  and  $\{V_2, V_3\}$  in the graph, then there exists 3-clique  $\{V_1, V_2, V_3\}$ . All 3-cliques are collected to the list  $C_3$ .

All  $(n + 1)$ -cliques can be constructed from the list  $C_n$ . Two  $n$ -cliques  $c_n^1$  and  $c_n^2$ , which have already  $(n - 1)$  nodes in common, are tested if they could form a new  $(n + 1)$ -clique  $c_{n+1}$ . There has to exist  $n$ -clique  $c_n^3$ , which has  $(n - 2)$  nodes in common with cliques  $c_n^1$  and  $c_n^2$ , in the list  $C_n$ . Additionally,  $(n - 1)$ th node of  $c_n^3$  has to be equivalent to  $n$ th node of  $c_n^1$  and  $n$ th node of  $c_n^3$  has to be equivalent to  $n$ th node of  $c_n^2$ , then there is  $(n + 1)$ -clique  $c_{n+1}$  in the graph. For example, if there exist cliques  $c_4^1 = \{V_1, V_2, V_3, V_4\}$ ,  $c_4^2 = \{V_1, V_2, V_3, V_5\}$  and  $c_4^3 = \{V_1, V_2, V_4, V_5\}$ , then there exist 5-clique  $c_5 = \{V_1, V_2, V_3, V_4, V_5\}$  in the graph. This procedure is repeated as long as new cliques can be constructed. All lists  $C_i, i = 1, \dots, n_{max}$  are tested in the end, that any  $n$ -clique is not a subclique of  $(n + m)$ -clique,  $m > 0$ . If there exist subcliques they can be eliminated. In the end, the dependent variable in all the cliques is selected such that the coefficient of determination is maximized.

The problem to find all maximal cliques is known to be *NP*-hard [14]. This means that the computational time for a solution is nondeterministic and the number of cliques can increase exponentially. Several other algorithms for solving the clique problem are introduced and analyzed in [14]. Computationally, the task is feasible, if the number of variables in the data  $D$  is not large. The number of variables can be a few hundred. The number of arcs in the graph also affects on the computational efficiency.

The number of found complete and maximal cliques can be large, but there are additional criteria how final cliques are selected. Firstly, two cliques can have only one variable or node in common. Secondly, the common variable cannot be a dependent variable in both cliques. Finally, cycles are not allowed in the dependency structure. Therefore, the dependency structure is a dependency tree or a forest under

these restrictions. The independent variables are the parents of the dependent variable in the final dependency structure. The construction of the dependency structure starts from the linear model which has the highest coefficient of determination. After that, the linear models are added such that the coefficients of determination are as good as possible and the previous restrictions are not violated.

## 5 Experiments

### 5.1 Data

A real-world data set which is used in this study is called the System data. The System data consist of nine measurements from a single computer which is connected to a network. The computer is used for example to edit programs or publications and to calculate computationally intensive tasks [23].

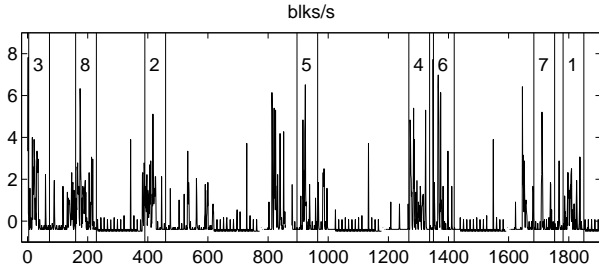
Four of the variables describe the network traffic. Rest of the variables are measurements from the central processing unit (CPU). All the variables are in relative measures in the data set. The variables are 1. `blks/s` (read blocks per second (network)), 2. `wblks/s` (written blocks per second (network)), 3. `usr` (time spent in user processes (CPU)), 4. `sys` (time spent in system processes (CPU)), 5. `intr` (time spent handling interrupts (CPU)), 6. `wio` (CPU was idle while waiting for I/O (CPU)), 7. `idle` (CPU was idle and not waiting for anything (CPU)), 8. `ipkts` (the number of input packets (network)), and 9. `opkts` (the number of output packets (network)).

The System data is collected during one week of computer operation. The first measurement is done in the morning on Monday and the last one is done in the evening on Friday. The measurements are done every two minutes during the day and every five minutes during the night. The measurements are done from every nine variables each time. There are missing values in all the variables. A more detailed description of the System data set is found from [23].

### 5.2 Preprocessing of the data

In general, processes are usually in different states during the measurements. It is possible that dissimilar linear dependency structures are needed to describe the operation of computer during the week, for example one structure during the day and another during the night. The variable `blks/s` can vary depending on if someone is working with the computer.

In this study, the similar states of the process are sought using the variable `blks/s`, which is, thus, a reference variable. The reference variable can be any of the variables in the data set depending on which feature is wanted to be explored. The reference variable is plotted in Figure 3.



**Figure 3. The reference variable  $\text{blks/s}$  and the selected windows.**

A query window is selected from the reference variable. The query window of reference variable should include information or the measurements of the feature i.e. the interesting state of the time-series, which is under exploration. The selected query window is the window number one in Figure 3. The measurements in the query window are done in the afternoon on Friday.

The similar states of the reference variable can be located mathematically in many ways. In this study, the sum of squares of differences between the query window and a candidate window is minimized. The candidate window is a part of the reference variable which is as long as the query window. The candidate windows are not allowed to overlap with each other or with the query window. The candidate windows which have the smallest sum of squares of differences between the query window are chosen.

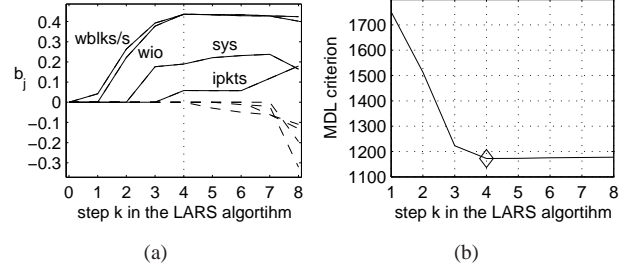
The sum of squares of differences between the query window and the candidate window i.e. the Euclidean distance between them is calculated as follows

$$E_c = \sum_{i=1}^M (y_{q,i} - y_{c,i})^2, \quad (9)$$

where  $y_q$  is the query window,  $y_c$  is the candidate window, and  $M$  is a number of the measurements which are included in the query window.

The number of chosen candidate windows can be decided, for example, by setting a threshold value to Equation (9). Another option is to select so many windows that there are enough data points in further calculations. In Figure 3, windows 2 – 8 are the chosen candidate windows. Smaller numbers of candidate windows refer to smaller values of Equation (9). The measurements in all the chosen candidate windows are done during the working hours.

The data in the chosen candidate windows and in the query window from the reference variable are chosen and the rest of the measurements are excluded from further calculations. The parts, which have the same time label as chosen candidate windows and query window, are also selected



**Figure 4. Development of coefficient values in the LARS algorithm (a). Values of MDL criterion for different models (b). The vertical line in (a) and the diamond in (b) represents the minimum value of MDL criterion.**

from the rest of the time-series. All the selected windows are scaled to have zero mean and unit variance. There are 70 data points in each selected window so in the further calculations there are  $N = 560$  data points in total from every variable.

New time-series are acquired when the selected windows of the original variables are put one after another. The original measurements often include noise. The level of the noise may be disturbingly high. In that case, noise reduction techniques can be applied to the selected windows, for example techniques based on the wavelet transform [5], [17].

This kind of similarity search in high dimensions, i.e. with very long time windows, should be approached with caution. The Euclidean distance between the windows may not work because of the curse of dimensionality. This selection of windows, however, is not central to this work, but it can be used if only the certain parts of the time-series are interesting and wanted to be explored.

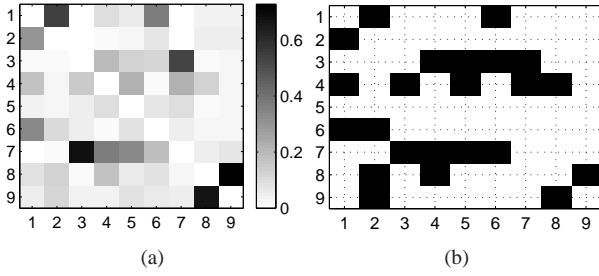
### 5.3 An example of sparse regression

The operation of LARS algorithm is illustrated with an example. The variable  $\text{blks/s}$  is the dependent variable and rest of the variables are the possible independent variables.

The independent variables are added to the regression model in the following order  $\text{wblks/s}$ ,  $\text{wio}$ ,  $\text{sys}$ ,  $\text{ipkts}$ ,  $\text{intr}$ ,  $\text{idle}$ ,  $\text{opkts}$ , and  $\text{usr}$ . Development of regression coefficients are plotted in the left panel of Figure 4. The values of MDL criterion are plotted in the right panel of the same figure. The minimum value is achieved by step four i.e. the first four added independent variables are included to the regression model.

The sparse regression model is

$$\hat{y}_{\text{blks/s}} = 0.44x_{\text{wblks/s}} + 0.45x_{\text{wio}} + 0.19x_{\text{sys}} + 0.07x_{\text{ipkts}}. \quad (10)$$



**Figure 5. The adjacency matrices of the belief graph  $G_b$  (a) and the unweighted directed graph  $G_d$  (b).**

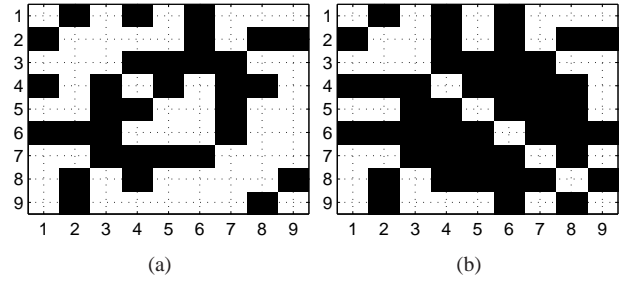
The coefficients of determination of the sparse model (10) and the full model are 0.89 and 0.90, respectively. Thus, the excluded variables can be considered as non-informative and dropping out them improve the interpretability of dependencies between the variables.

#### 5.4 The dependency structure of System data

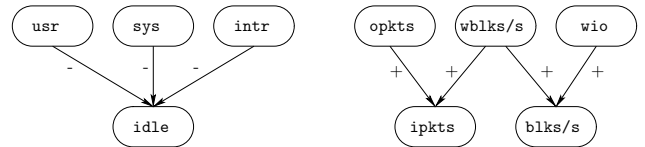
The objective is to find the best multiple linear regression models from the preprocessed data set. The process proceeds as it is described in Section 4. The first task is to construct the belief graph  $G_b$ .

The adjacency matrix of the belief graph  $G_b$  is presented in the left panel of Figure 5. The relative weights of the regression coefficients of the  $i$ th model are in the  $i$ th column in the adjacency matrix. The  $i$ th variable has been the dependent variable in the  $i$ th model and rest of the variables have been the possible independent variables. The relative weights for the variables 2, . . . , 8, when the variable 1 is the dependent variable, are presented in the first column of the adjacency matrix of  $G_b$ . The other columns are constructed in a corresponding way. Dark colors refer to a strong belief that these variables are significant in the multiple linear regression model. For example, in the first column or in the first regression model the variables 2 (*wblks/s*), 4 (*sys*), and 6 (*wio*) are the most significant independent variables. The number of bootstrap replications was  $B = 1000$  in each of the nine cases. The relative weights of the coefficients were calculated according to Equation (8).

The directed graph  $G_d$  is computed from  $G_b$  using the threshold  $\lambda = 0.1$  i.e the dependencies whose relative weight is under 0.1 are ignored and rest of the weights are set to unity. The adjacency matrix of  $G_d$  is in the right panel of Figure 5. The unweighted undirected graph  $G_u$  is obtained from  $G_d$  by ignoring the directions of dependencies and the moral graph  $G_m$  is calculated from  $G_u$  as it is described in Section 4.2. The adjacency matrices of  $G_u$  and  $G_m$  are drawn in Figure 6.



**Figure 6. The adjacency matrices of the unweighted undirected graph  $G_u$  (a) and the moral graph  $G_m$  (b).**



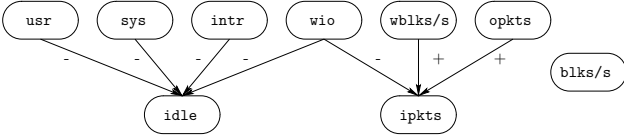
**Figure 7. The dependency forest from the graph  $G_u$ .**

The final linear models are sought from the undirected graph  $G_u$  and from the moral graph  $G_m$ . The variables which belong to the same multiple linear regression model are part of the same maximal clique in the graphs  $G_u$  or  $G_m$ . The maximal cliques are found using the algorithm which is presented in Section 4.3.

Three maximal cliques  $c_4^1 = \{\text{idle}, \text{usr}, \text{sys}, \text{intr}\}$ ,  $c_3^1 = \{\text{ipkts}, \text{opkts}, \text{wblks/s}\}$  and  $c_3^2 = \{\text{blks/s}, \text{wblks/s}, \text{wio}\}$  were found from the graph  $G_u$ . The best models are achieved if the variables *idle*, *ipkts* and *blks/s* are chosen to be the dependent variables. The coefficients of determination are then 0.95, 0.94 and 0.82. The dependency forest of these linear models is in Figure 7.

All variables in clique  $c_4^1$  are measurements from the CPU. All regression coefficients were negative in this model. If there is a positive change in some independent variable, the value of the dependent variable *idle* will decrease.

Cliques  $c_3^1$  and  $c_3^2$  are dependent on each other through the variable *wblks/s*, which is one of the independent variables in both models. When a positive change occurs in the variable *wblks/s* also the values of the dependent variables *ipkts* and *blks/s* increase. All variables in the clique  $c_3^1$  are the measurements from the network traffic. In the clique  $c_3^2$ , the variable *blks/s* is the measurement from the network traffic and the variable *wio* is the measurement from the CPU.



**Figure 8. The dependency tree from the graph  $G_m$ .**

An alternative dependency structure is computed from the graph  $G_m$ . There were two maximal cliques and the variable `blks/s` was left alone. Cliques are  $c_5^1 = \{\text{idle}, \text{usr}, \text{sys}, \text{intr}, \text{wio}\}$  and  $c_4^1 = \{\text{ipkts}, \text{wio}, \text{wblks/s}, \text{opkts}\}$ . The dependency structure is plotted in Figure 8.

All the variables in the clique  $c_5^1$  are measurements from the CPU. The best model is obtained, when `idle` is the dependent variable. Then the coefficient of determination is 0.96, which indicates that the linear model describes the dependencies between the variables very well. The first linear sparse regression model is

$$\hat{y}_{\text{idle}} = -0.64x_{\text{usr}} - 0.30x_{\text{sys}} - 0.13x_{\text{intr}} - 0.08x_{\text{wio}}. \quad (11)$$

The independent variables describe how much of the CPU power is spent to different activities and the dependent variable describes how much of the CPU power is unused at the moment. According to the estimated linear model the value of `idle` decreases when the values of `usr`, `sys`, `intr` and `wio` increase. This is very intuitive result because the processes which need CPU power obviously diminish the available CPU power.

The clique  $c_4^1$  formulates another multiple linear regression model. When the variable `ipkts` is selected to the dependent variable, the best coefficient of determination (0.94) is achieved. The second model is

$$\hat{y}_{\text{ipkts}} = -0.01x_{\text{wio}} + 0.11x_{\text{wblks/s}} + 0.93x_{\text{opkts}}. \quad (12)$$

The variable `ipkts` consists of measurements from the network traffic. The variables `wblks/s` and `opkts` describes also the network traffic and `wio` is the same measurement from the CPU as in model (11). When the number of written blocks per second and the number of output packets increase the number of input packets also increases according to Equation (12). This is a natural situation in the bidirectional network traffic. The packets are sent to both directions when for example a file is downloaded.

Models (11) and (12) are dependent on each other through the variable `wio`. Changes in `wio` has effect on both dependent variables `idle` and `ipkts`. A positive change in `wio` decreases the values of `idle` and `ipkts`. However, the variable `wio` could be possibly excluded from model (12), since the value of its regression coefficient is negligible and it has not much effect on the coefficient of determination.

## 6 Summary and conclusion

In this study, the method for analyzing linear dependencies in multivariate data is proposed. The result of the method is a linear dependency tree or a forest. The dependencies of the variables can be clearly seen from the final dependency structure. Thus, it is possible to get deeper understanding of the underlying process. The linear dependencies are modeled by multiple linear regression models.

Similar states of time-series are selected using the Euclidean distance between a reference variable. A single regression model is constructed to model that selected state. It may be difficult or even impossible to construct a single regression model to time-series, which consists of many different states. Every state would require a model of its own.

This study proposes how the relative weights of the regression coefficients can be calculated from the bootstrap replications. The relative weight of the regression coefficient is a measure of belief that the corresponding independent variable belongs to a certain regression model. In addition, the dependent variable or variables are selected during the execution of the algorithm.

In the experiments it was shown that the most significant variables have the highest relative weights. The relative weights seem to be appropriate to measure significance of the independent variables.

The final dependency structure was constructed from the belief graph. The belief graph represents the variables and the relative strength of the dependencies between the variables. A threshold value was used to reduce the dependencies in the belief graph. The chosen threshold value has a strong impact on the final dependency structure. A minor change in the threshold value can cause a major changes in the final dependency structure. Thus, special attention to the threshold value should be paid. It would be beneficial to automate the selection of the threshold value. One possibility might be to include it somehow in the moralizing operation. Another possibility could be to learn the final dependency structure from the belief graph ignoring the weakest dependencies by some data based method such that a tree or a forest structure is achieved.

The proposed method was tested using a real world-data set. The constructed dependency structures were convincing. Approximately 95% of the variation of dependent vari-

ables was explained by the regression models which were constructed from the System data, although no assumptions were made about the number of linear models. The resulting dependency structure was almost similar to one shown in Figure 8, when the whole data set was used in the construction of the dependency structure. When models (11) and (12) were tested with the excluded data the coefficients of determination were nearly as good as with the used data.

The final dependency structure highlights the dependencies of the variables in an intelligible way. On the other hand, it is difficult to measure the level of interpretability. The goodness of the models may be hard to justify if coefficient of determinations are moderate, but the dependency structure can still give additional and unexpected information in co-operation with someone who has specific knowledge of the underlying process.

## 7 Acknowledgements

The authors thank Dr. Esa Alhoniemi for the idea of selecting similar windows from time-series. We also thank for the insightful comments from the reviewers of the paper.

## References

- [1] L. Breiman. The little bootstrap and other methods for dimensionality selection in regression: X-fixed prediction error. *Journal of the American Statistical Association*, 87(419):738–754, September 1992.
- [2] L. Breiman. Better subset regression using the nonnegative garrote. *Technometrics*, 37(4):373–384, November 1995.
- [3] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, May 1968.
- [4] J. Copas. Regression, prediction and shrinkage. *Journal of the Royal Statistical Society (Series B)*, 45(3):311–354, 1983.
- [5] D. L. Donoho. De-noising by soft-thresholding. *IEEE Transactions on Information Theory*, 41(3):613–627, May 1995.
- [6] B. Efron. Bootstrap methods: Another look at the Jackknife. *The Annals of Statistics*, 7(1):1–26, January 1979.
- [7] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407–499, April 2004.
- [8] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, 1993.
- [9] M. H. Hansen and B. Yu. Model selection and the principle of minimum description length. *Journal of the American Statistical Association*, 96(454):746–774, June 2001.
- [10] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.
- [11] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, October 2000.
- [12] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, February 1970.
- [13] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15(3):225–263, October 1996.
- [14] I. Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science*, 250(1-2):1–30, January 2001.
- [15] F. Kose, W. Weckwerth, T. Linke, and O. Fiehn. Visualizing plant metabolomic correlation networks using clique-metabolite matrices. *Bioinformatics*, 17(12):1198–1208, December 2001.
- [16] K. Lagus, E. Alhoniemi, and H. Valpola. Independent variable group analysis. In G. Dorffner, H. Bischof, and K. Hornik, editors, *Proceedings of the International Conference on Artificial Neural Networks*, number 2130 in Lecture Notes in Computer Science, pages 203–210. Springer, 2001.
- [17] M. Lang, H. Guo, J. E. Odegard, C. S. Burrus, and R. O. Wells. Noise reduction using an undecimated discrete wavelet transform. *IEEE Signal Processing Letters*, 3(1):10–12, January 1996.
- [18] C. L. Mallows. Some comments on  $C_p$ . *Technometrics*, 15(4):661–675, November 1973.
- [19] C. L. Mallows. More comments on  $C_p$ . *Technometrics*, 37(4):362–372, November 1995.
- [20] G. M. Maruyama. *Basics of Structural Equation Modeling*. SAGE Publications, Inc., 1997.
- [21] P. Stoica and Y. Selen. Model-order selection. *IEEE Signal Processing Magazine*, 21(4):36–47, July 2004.
- [22] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.
- [23] J. Vesanto and J. Hollmén. An automated report generation tool for the data understanding phase. In A. Abraham, L. Jain, and B. J. van der Zwaag, editors, *Innovations in Intelligent Systems: Design, Management and Applications*, Studies in Fuzziness and Soft Computing. Springer (Physica) Verlag, 2003.

# Unsupervised Learning of Sequential Patterns

Andreas D. Lattner and Otthein Herzog  
TZI – Center for Computing Technologies  
Universität Bremen, PO Box 330 440,  
28334 Bremen, Germany  
[adl|herzog]@tzi.de

## Abstract

*In order to allow agents for acting autonomously and making their decisions on a solid basis an interpretation of the current scene has to be done. Scene interpretation can be done by checking if certain patterns match to the current belief of the world. In some cases it is impossible to acquire all situations an agent might have to deal with later at run-time in advance. Here, an automated acquisition of patterns would help an agent to adapt to the environment. Agents in dynamic environments have to deal with world representations that change over time. Additionally, predicates between arbitrary objects can exist in their belief of the world. In this work we present a learning approach that learns temporal patterns from a sequence of predicates.*

## 1. Introduction

In order to allow agents for acting autonomously and making their decisions on a solid basis, an interpretation of the current scene has to be done. Scene interpretation can be done by checking if certain patterns match to the current belief of the world. If intentions of other agents or events that are likely to happen in the future can be recognized the agent's performance can be improved as it can adapt the behavior to the situation. Example domains are intelligent vehicles or robots in the RoboCup leagues [10, 11].

In some cases it is impossible or too time-consuming to acquire all situations an agent might have to deal with at run-time in advance, or it is necessary that the agent adapts to a certain situation. It might even not be known what situations an agent will encounter in the future. In these cases an automated acquisition of patterns would help an agent to adapt to the environment.

We focus on qualitative representations as they allow for a concise representation of the important information. Such a representation allows to use background knowledge, to plan future actions, to recognize plans of other agents, and

is comprehensible for humans the same time. In our approach we map quantitative data to qualitative representations. Time series are divided into different segments which satisfy certain monotonicity or threshold conditions as it is suggested by Miene et al. [10, 11]. E.g., if the distance between two objects is observed it can be divided into increasing and decreasing distance representing approaching and departing relations (cf. [11]).

Agents in dynamic environments have to deal with world representations that change over time. Additionally, predicates between arbitrary objects can exist in their belief of the world. Current learning approaches do not handle these requirements properly. In this work we present a learning approach that learns temporal patterns from a sequence of predicates.

The paper is organized as follows: Section 2 presents work related to ours. The representation of scenes and patterns is described in Sections 3 and 4. Section 5 addresses the learning of patterns. A first evaluation of our approach is presented in Section 6. The paper closes with a conclusion and some ideas for future work.

## 2. Related Work

There are many approaches in the areas of association rule mining, inductive logic programming, and spatial, temporal, or spatiotemporal data mining which are relevant to the work presented here. The most important ones are presented in this section.

Association rule mining addresses the problem of discovering association rules in data. One famous example is the mining of rules in basket data [1]. Agrawal et al. and Zaki et al. present fast algorithms for mining association rules [2, 14]. These approaches have been developed for the mining of association rules in item sets. Thus, no temporal relationships between the items are represented.

Mannila et al. extended association rule mining by taking event sequences into account [8]. They describe al-

gorithms which find all relevant episodes which occur frequently in the event sequence.

Höppner presents an approach for learning rules about temporal relationships between labeled time intervals [5]. The labeled time intervals consist of propositions. Similar to our work interval relationships are described by Allen’s interval logic [3]. The major difference is that predicates which represent relations between certain objects are not considered in their work.

Other researchers in the area of spatial association rule mining allow for more complex representations with variables but do not take temporal interval relations into account (e.g., [6, 7, 9]).

The work presented here combines ideas from different directions. Similar to Höppner’s work [5] the learned patterns describe temporal interrelationships with Allen’s interval logic. Contrary to Höppner’s approach our representation allows for describing predicates between different objects similar to approaches like [7]. The generation of frequent patterns comprises a top-down approach starting from the most general pattern and specializing it. At each level of the pattern mining just the frequent patterns of the previous step are taken into account knowing that only combinations of frequent patterns can result in frequent patterns again. This is a typical approach in association rule mining (e.g., [8]).

Tan et al. present different interestingness measures for association patterns [13]. In our work some new evaluation criteria for the more complex representation are defined.

### 3. Scene Representation

A dynamic scene is represented by a set of objects and predicates between these objects. The predicates are only valid for certain time intervals and the scene can thus be considered as a sequence of (spatial or conceptual) predicates. These predicates are in specific temporal relations regarding the time dimension. Miene et al. presented an approach how to create such a representation [11].

Each predicate  $r$  is an instance of a predicate definition  $rd$ . We use the letter  $r$  for predicates/relations; the letter  $p$  is used for patterns.  $\mathcal{R}_{schema} = \{rd_1, rd_2, \dots\}$  is the set of all predicate definitions  $rd_i := \langle l_i, a_i \rangle$  with label  $l_i$  and arity  $a_i$ , i.e., each  $rd_i$  defines a predicate between  $a_i$  objects<sup>1</sup>. Predicates can be hierarchically structured. If a predicate definition  $rd_1$  specializes another predicate definition  $rd_2$  all instances of  $rd_1$  are also instances of the super predicate  $rd_2$ .

<sup>1</sup>It would also be possible to specify more formally what objects are including instance-of relations, attributes, etc.; this is omitted here in order to reduce complexity for the moment. So far we just see objects as uniquely identifiable entities.

If we have to handle more than one dynamic scene, let  $S = \{s_1, s_2, \dots\}$  be the set of the different sequences  $s_i$ . A single sequence  $s_i$  is defined as

$$s_i = (\mathcal{R}_i, \mathcal{TR}_i, \mathcal{C}_i)$$

where  $\mathcal{R}_i$  is the set of predicates,  $\mathcal{TR}_i$  is the set of temporal relations and  $\mathcal{C}_i$  is the set of constants representing different objects in the scene. Each predicate is defined as  $r(c_1, \dots, c_n)$  with  $r$  being an instance of  $rd_i \in \mathcal{R}_{schema}$ , having arity  $n = a_i$ , and  $c_{i,1}, \dots, c_{i,n} \in \mathcal{C}_i$  are representing the objects where the predicate holds. The set of temporal relations  $\mathcal{TR}_i = \{tr_1, tr_2, \dots\}$  defines relations between pairs of elements in  $\mathcal{R}_i$ . Each temporal relation is defined as  $tr_i(r_a, op, r_b)$  with  $r_a, r_b \in \mathcal{R}_i$  and  $op \in \{<, =, >, d, di, o, oi, m, mi, s, si, f, fi\}$ , i.e., Allen’s temporal relations between intervals [3].

Note that it can happen that the same set of objects appears in different instances of an identical type of predicate at different times. These predicates must be represented by different predicate instances as they usually do not have identical temporal interrelations with other predicates, e.g., two vehicles which are in the relation `faster(v1, v2)` in two different time intervals  $i_1$  and  $i_2$ . Therefore, we assign IDs  $r_i$  to the different predicates. An example is:

```
r1 = behind(obj1, obj2)
r2 = accelerates(obj1)
r3 = leftOf(obj2, obj3)
tr1 = r1 o r2
tr2 = r1 < r3
tr3 = r2 < r3
```

Here, three predicates exist between `obj1`, `obj2`, and `obj3` (`behind`, `accelerates`, and `leftOf`). These predicates have certain temporal restrictions: `r1` must be before `r3`, `r2` must be before `r3`, and `r1` overlaps `r2`.

### 4. Pattern Description

This section introduces the representation of patterns, how patterns are to be matched in sequences, and how patterns can be specialized or generalized.

#### 4.1. Pattern Representation

Patterns are abstract descriptions of sequence parts with specific properties. A pattern defines what predicates must occur and how their temporal interrelationship has to be. Let  $\mathcal{P} = \{p_1, p_2, \dots\}$  be the set of all patterns  $p_i$ . A pattern is (similar to sequences) defined as

$$p_i = (\mathcal{R}_i, \mathcal{TR}_i, \mathcal{V}_i).$$

$\mathcal{R}_i$  is the set of predicates  $r_{ij}(v_{ij,1}, \dots, v_{ij,n})$  with  $v_{ij,1}, \dots, v_{ij,n} \in \mathcal{V}_i$ .  $\mathcal{V}_i$  is the set of all variables used in

the pattern.  $\mathcal{TR}_i$  defines the set of the temporal relations which have already been defined above.

An example for such a pattern description is the following abstraction of the sequence above (capital letters stand for variables, lower case letters for constants):

```
r1 = accelerates(X)
r2 = behind(X, Y)
tr1 = r1 o r2
```

In this pattern there are two predicates and  $r1$  overlaps  $r2$ .

## 4.2. Pattern Matching

A pattern  $p$  matches in a (part of a) sequence  $sp$  if there exists a mapping of a subset of the constants in  $sp$  to all variables in  $p$  such that all predicates defined in the pattern exist between the mapped objects and all time constraints of  $p$  are satisfied by the time intervals in the sequence. In order to restrict the exploration region a window size can be defined. Only matches within a certain neighborhood (specified by the window size) are valid. In our case the window size determines the number of adjacent start and end time points of different predicates which are taken into account for matching a pattern.

During the pattern matching algorithm a sliding window is used, and at each position of the window all matches for the different patterns are collected. A match consists of the position in the sequence and an assignment of objects to the variables of the pattern.

## 4.3. Specialization and Generalization of Patterns

Different patterns can be put into generalization-specialization relations. A pattern  $p_1$  subsumes another pattern  $p_2$  if it covers all sequence parts which are covered by  $p_2$ :

$$p_1 \sqsubseteq p_2 := \forall sp, matches(p_2, sp) : matches(p_1, sp).$$

If  $p_1$  additionally covers at least one sequence part which is not covered by  $p_2$  it is more general:

$$p_1 \sqsubset p_2 := p_1 \sqsubseteq p_2 \wedge \exists sp_x : matches(p_1, sp_x), \neg matches(p_2, sp_x).$$

This is the case if  $p_1 \sqsubseteq p_2 \wedge p_1 \not\sqsubseteq p_2$ .

In order to specialize a pattern it is possible to add a new predicate  $r$  to  $\mathcal{R}_i$ , add a new temporal relation  $tr$  to  $\mathcal{TR}_i$ , unify two variables, or specialize a predicate, i.e., replacing it with another more special predicate. Accordingly it is possible to generalize a pattern by removing a predicate  $r$  from  $\mathcal{R}_i$ , removing a temporal relation  $tr$  from  $\mathcal{TR}_i$ , inserting a new variable, or generalizing a predicate  $r$ , i.e., replacing it with another more general predicate.

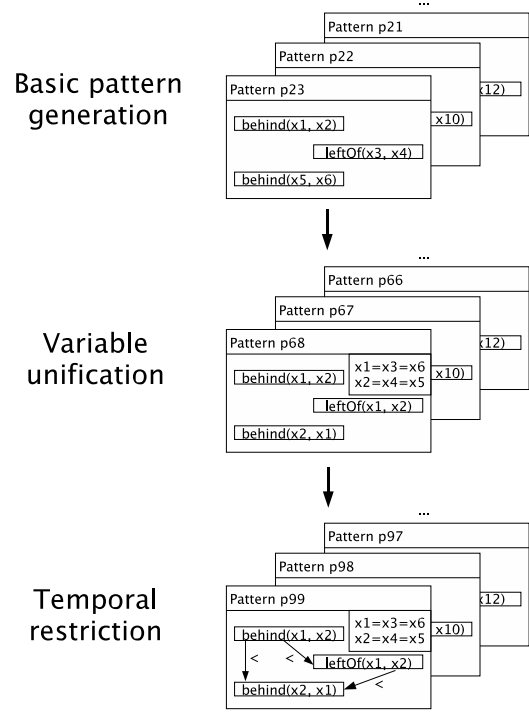


Figure 1. Pattern generation

At the specialization of a pattern by adding a predicate a new instance of any of the predicate definitions can be added to the pattern with variables which are not used in the pattern so far. If a pattern is specialized by adding a new temporal relation for any pair of predicates in the pattern (which has not been constrained so far) a new temporal restriction can be added. A specialization through variable unification can be done by unifying an arbitrary pair of variables. Another specialization would be to replace a predicate by a more special predicate. These different specialization steps can be used at the top-down induction of rules as described in Section 5.1.

## 5. Unsupervised Learning of Sequential Patterns

Unsupervised learning of patterns in a sequence  $S$  (or in a set of sequences  $S$ ) is more difficult than learning from examples as there is no clear task to separate positive from negative examples. The task is to find “interesting” patterns which appear repeatedly. Unless there is no more information how to identify the patterns of interest, it just can be searched for patterns that appear frequently in the sequence of data.

The following subsections present the learning algorithm and evaluation criteria how to estimate the quality of a pattern.

## 5.1. Top-down Induction

In order to create frequent patterns top-down induction is applied. In the first step all frequent patterns of the desired pattern size (i.e., number of used predicates) are created. Here, for all predicates new variables are created. In the second step, the patterns are specialized by the unification of variables, i.e., the different predicates can be “connected” via identical variables after this step. In the third step temporal restrictions are added to the patterns. If a maximum value for the number of patterns to keep (*maxPatterns*) is set, after each step only the best patterns are kept. The basic steps are illustrated in Figure 1. The pattern generation algorithm is:

### 1. Basic pattern generation.

- (a) Create initial patterns (consisting of just one predicate each) and store them in the *currentPatternsList*. Set *currentLevel* to 1.
- (b) Remove all patterns from the *currentPatternsList* which do not satisfy certain conditions (e.g., minimum frequency). Add all patterns in *currentPatternsList* to *allPatternsList* if *currentLevel* < *minPatternLevel*.
- (c) If *currentLevel* ≤ *maxPatternLevel* create next level patterns: Clear *currentPatternsList*, specialize all frequent patterns by combining them with all predicates of the frequent patterns of the previous step and store them in the *currentPatternsList*. Keep the best *maxPatterns* patterns in the *allPatternsList*. Go to step 1b.

### 2. Specialize by variable unification.

- (a) Set the *currentPatternList* to *allPatternList*.
- (b) For each pattern in *currentPatternList* create all possible variable unifications and store them in the cleared *currentPatternList*.
- (c) Remove all patterns from the *currentPatternsList* which do not satisfy certain conditions (e.g., minimum frequency). Add all patterns in *currentPatternsList* to *allPatternsList*.
- (d) If new patterns were created go to step 2b.
- (e) Keep the best *maxPatterns* patterns in the *allPatternsList*.

### 3. Specialize by temporal relations.

- (a) Set the *currentPatternList* to *allPatternList*.
- (b) For each pattern create statistics about temporal relations between predicates (i.e., count the occurrences of the different interval relations for each pattern pair).
- (c) If the creation of a temporal relation between two predicates in a pattern does not violate certain conditions (e.g., minimum frequency), the pattern is copied, specialized by this temporal relation, and added to the cleared *currentPatternList*.
- (d) Add all patterns in *currentPatternsList* to *allPatternsList*. If new patterns were created go to step 3b.
- (e) Keep the best *maxPatterns* patterns in the *allPatternsList*.

The parameters of the algorithm are:

- **minPatternLevel:** The minimum size of the patterns (i.e., the minimum number of predicates appearing in each pattern).
- **maxPatternLevel:** The maximum size of the patterns (i.e., the maximum number of predicates appearing in each pattern).
- **maxPatterns:** The maximum number of patterns to be kept after each step of the algorithm.

## 5.2. Pattern Evaluation

In order to evaluate the patterns (and to keep the *k* best patterns) an evaluation function was defined. Right now our evaluation function takes five different values into account: relative pattern size, relative pattern frequency, coherence, temporal restrictiveness, and predicate preference. Further criteria can be added if necessary.

The relative pattern size is the pattern size divided by the maximum pattern size:

$$relPatSize = \frac{patSize}{maxPatSize}$$

This measure prefers larger patterns. The assumption here is that patterns with more predicates which are still frequent contain more useful information than shorter ones.

The relative pattern frequency is the number of matches multiplied by the pattern size relative to the length of the whole sequence:

$$relPatFrq = \frac{patSize \cdot numMatches}{sequenceLength}$$

Patterns which appear more often are assigned a higher value.

The coherence gives information how “connected” the predicates in the pattern are by putting the number of used variables in relation to the maximum possible number of variables for a pattern:

$$coh = 1 - \frac{numVariables}{maxNumberVariables}$$

Here, the idea is that patterns which are highly connected through variables provide more information because they are interrelated by shared objects. Currently we just regard binary predicates, thus

$$maxNumberVariables = 2 \cdot patSize.$$

The temporal restrictiveness is the number of restricted predicate pairs in the pattern to the maximum number of time restrictions:

$$tmpRestr = \frac{numRestrictedPredPairs}{maxNumRestrictedPredPairs} = \frac{numRestrictedPredPairs}{\frac{patSize \cdot (patSize - 1)}{2}}$$

Again it is assumed to be advantageous if the pattern is more specialized. A higher number of temporal restrictions leads to a higher value.

Predicate preference computes the mean of all used predicate preferences:

$$prPref = \frac{1}{patSize} \sum_{i=1}^{patSize} preference(predicate_i)$$

This is needed if certain predicates should be privileged, e.g., if some predicates are assumed to be more useful than others.

The overall evaluation function of a pattern computes a weighted sum of the five values:

$$eval = \frac{a \cdot relPatSize + b \cdot relPatFrq + c \cdot coh + d \cdot tmpRestr + e \cdot prPref}{a + b + c + d + e}$$

All single measures and the overall pattern quality have values between 0 and 1.

## 6. Evaluation

This section presents a first evaluation of our learning algorithm. First, the idea is shown by applying the algorithm to a simple example. After that experiments on some more complex data with varying parameters are presented. Finally, some statements about the complexity of the current algorithm are given.

```
s; behind;      a; b           e; approaches; x; y
s; faster;     x; y           s; behind;     c; a
e; behind;     a; b           e; behind;     c; a
s; leftOf;     a; b           s; behind;     a; d
e; leftOf;     a; b           s; follows;    x; y
e; faster;     x; y           e; behind;     a; d
s; behind;     b; a           s; leftOf;     a; d
e; behind;     b; a           e; leftOf;     a; d
s; behind;     a; c           e; follows;    x; y
s; approaches; x; y           s; behind;     d; a
e; behind;     a; c           e; behind;     d; a
s; leftOf;     a; c           s; faster;     x; z
e; leftOf;     a; c           e; faster;     x; z
```

Figure 2. Input for sequence example

Step	New patterns	Kept patterns	All patterns
Initial patterns	5	5	0
Pattern level 2	15	10	10
Pattern level 3	28	13	23
Unification 1	255	57	80
Unification 2	393	51	131
Unification 3	200	15	146
Unification 4	32	1	147
Unification 5	1	0	147
Temp. Rest. 1	13	13	160
Temp. Rest. 2	21	21	181
Temp. Rest. 3	9	9	190

Table 1. Number of patterns in the different steps of the algorithm

### 6.1. Simple Example

In this subsection patterns for a simple artificial example are created. In the example there are seven different objects (a - d and x - z), five predicates (behind, faster, leftOf, approaches, and follows). The input data is shown in Figure 2. The first column gives the information if the interval starts or ends in this row. The names of the predicates are in the second column. The third and fourth columns show the IDs of the objects for which the predicate holds. The temporal order is represented by the rows from top to bottom.

Figure 3 shows the different intervals in this example. As it can be seen, object a has behind and leftOf relations to the objects b, c, and d. The predicates with x, y, and z are not important; they are just “noise” in the example.

The program was started with these parameters:

- $minPatternSize = 2$ ,
- $maxPatternSize = 3$ , and
- $maxPatterns = \infty$  (i.e., keep all created patterns).

The weights of the evaluation function were set to:  $a = 0.2$ ,  $b = 0.4$ ,  $c = 0.2$ ,  $d = 0.2$ , and  $e = 0.0$ . The predicate preference has not been realized so far and is thus not taken into account here.

Table 1 shows the number of patterns for the different steps. Note that patterns are just kept if they have more than

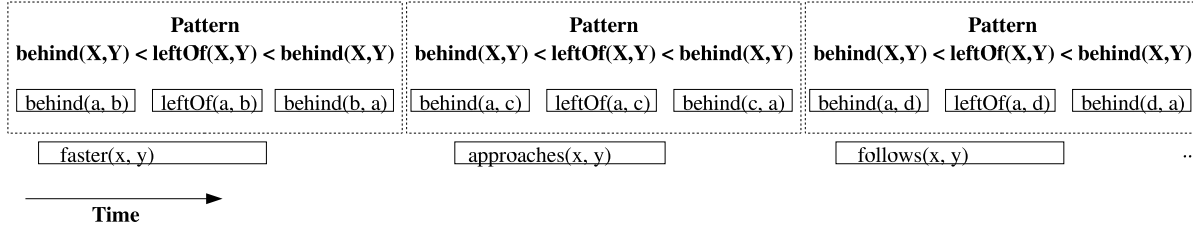


Figure 3. Example sequence and matched pattern

Pattern	Overall quality
behind(X <sub>9</sub> X <sub>29</sub> )[r <sub>1</sub> ], leftOf(X <sub>9</sub> X <sub>29</sub> )[r <sub>2</sub> ], behind(X <sub>29</sub> X <sub>9</sub> )[r <sub>3</sub> ], r <sub>1</sub> <->r <sub>2</sub> : before, r <sub>1</sub> <->r <sub>3</sub> : before, r <sub>2</sub> <->r <sub>3</sub> : before	0.780
behind(X <sub>9</sub> X <sub>29</sub> )[r <sub>1</sub> ], leftOf(X <sub>9</sub> X <sub>29</sub> )[r <sub>2</sub> ], behind(X <sub>29</sub> X <sub>9</sub> )[r <sub>3</sub> ], r <sub>1</sub> <->r <sub>2</sub> : before, r <sub>1</sub> <->r <sub>3</sub> : before	0.713
behind(X <sub>9</sub> X <sub>29</sub> )[r <sub>1</sub> ], leftOf(X <sub>9</sub> X <sub>29</sub> )[r <sub>2</sub> ], behind(X <sub>29</sub> X <sub>9</sub> )[r <sub>3</sub> ], r <sub>1</sub> <->r <sub>3</sub> : before, r <sub>2</sub> <->r <sub>3</sub> : before	0.713
behind(X <sub>9</sub> X <sub>29</sub> )[r <sub>1</sub> ], leftOf(X <sub>9</sub> X <sub>29</sub> )[r <sub>2</sub> ], behind(X <sub>29</sub> X <sub>9</sub> )[r <sub>3</sub> ], r <sub>1</sub> <->r <sub>2</sub> : before, r <sub>2</sub> <->r <sub>3</sub> : before	0.713
behind(X <sub>9</sub> X <sub>29</sub> )[r <sub>1</sub> ], leftOf(X <sub>9</sub> X <sub>29</sub> )[r <sub>2</sub> ], behind(X <sub>29</sub> X <sub>9</sub> )[r <sub>3</sub> ], r <sub>2</sub> <->r <sub>3</sub> : before	0.646

Table 2. Five best patterns

one predicate. The five best patterns are shown in Table 2. The intended pattern in the example was a simplified overtaking scenario (first, the overtaking car is behind another car, then it is left of, and finally the objects in the behind predicate are switched). As it can be seen, the intended pattern was found and evaluated best. Matched sequence parts are illustrated in Figure 3, enclosed by a dashed line.

## 6.2. Experiments

We evaluated the algorithm on an artificial data set with different settings. In order to create random data sets we wrote a program which generates input data for our learning program. Different parameters allow for creating different test sets. The parameters are: number of predicates, number of constants, fraction of “real patterns” in the random data, and the length of the sequence. As the “real pattern” the one above ( $\text{behind}(X, Y)$ ,  $\text{leftOf}(X, Y)$ ,  $\text{behind}(Y, X)$ ) was used with varying constants for  $X$  and  $Y$  and some random predicates inside the pattern.

For the experiments random sequences with 500 predicates (i.e., 1000 start and end time points) were created. The evaluation function was the same as shown above in Section 6.1. The window size was set to 15 (large enough to find the intended pattern). The number of constants (representing objects) was five in all settings. The minimum frequency in order to keep a pattern was 0.1. Figure 4 shows the number of created and kept patterns for the different settings in the experiments.

The number of predicates was varied in the first experi-

ment (5, 7, 9, and 11). One interesting fact is that with an increasing number of predicates the number of created patterns decreases. This seems counterintuitive at first glance. This phenomenon can be explained by the randomly created data. The more predicates are used the less frequent patterns with (random) predicates are created. This explains the decreasing number of created patterns in the first experiment.

In the second experiment the maximum size of the patterns was set to different values (2 - 5). This experiment shows the problem of complexity. With the increasing maximal size of patterns the number of created patterns grows exponentially. Future modifications of the algorithm should address this problem.

In the third experiment the number of patterns to keep after each step of the algorithm was varied ( $\text{maxPatterns} = 5, 10, 15, 20$ ). It shows how the number of created (and kept) patterns can be reduced by just keeping a certain number of patterns after each step of the basic pattern generation. The number of kept patterns almost stays constant. This might be due to the fact that many of the bad performing patterns are not created as their “parents” are pruned early while the important patterns are kept after each step.

## 6.3. Complexity

The algorithm consists of three main steps. The basic pattern generation algorithm creates predicate combinations up to an upper bound (the pattern size). If all predicate combinations were created the number of created patterns would be equal to the number of possibilities to take  $k$  elements out of  $n$  elements ignoring the order and allowing to draw elements repeatedly. Let  $n$  be the number of predicates and  $k$  the maximum pattern size. Then the number of basic patterns to be created is:

$$\#basicPatterns = \binom{n+k-1}{k}$$

Table 3 shows the number of possible patterns for different pattern sizes if five predicates exist, i.e.,  $n = 5$ .

The variable unification problem is similar to the problem of creating all possible clusters of arbitrary sizes for  $n$  objects. This leads to the Bell number [4]. The number

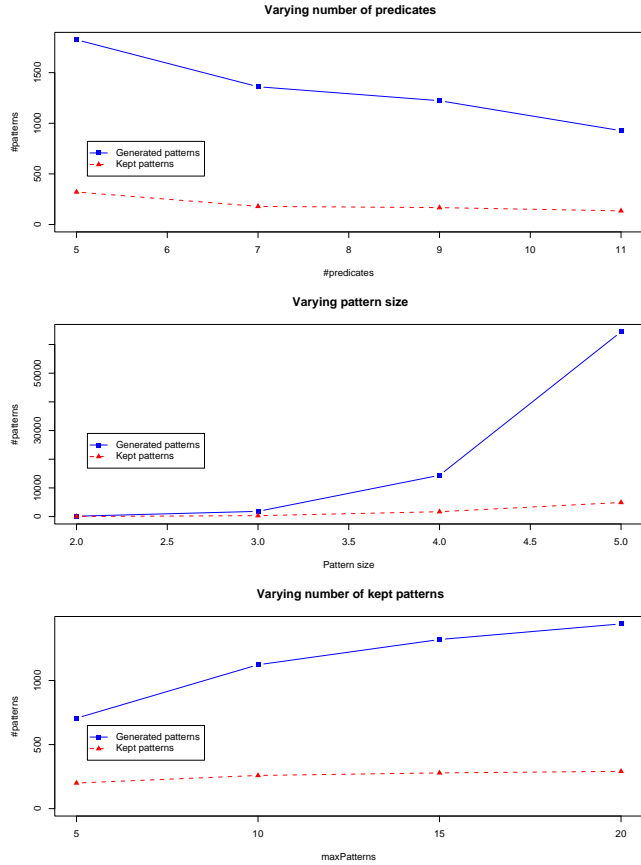


Figure 4. Evaluation results

Size of pattern	1	2	3	4	5
#possiblePatterns	5	15	35	70	126

Table 3. Number of possible patterns with five predicates

of all possible variable combinations for  $n$  variables can be computed by the formula (cf. [12]):

$$\begin{aligned} \#posVarUnific &= \sum_{k=1}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} \\ &= \sum_{k=1}^n \frac{1}{k!} \sum_{j=1}^k (-1)^{k-j} \binom{k}{j} j^n \end{aligned}$$

The Bell numbers for different  $n$  are shown in Table 4.

The number of possible temporal restrictions for a pattern depends on the number of predicates used in the pattern

n	1	2	3	4	5	6	7	8	9	10
$B_n$	1	2	5	15	52	203	877	4140	21147	115975

Table 4. Bell numbers

Size of pattern	1	2	3	4	5
#possibleRestrictions	1	14	$14^3$	$14^6$	$14^{10}$

Table 5. Number of possible temporal restrictions with five predicates

and the number of temporal relations. It is possible to assign one of Allen’s 13 relations or no relation to each predicate pair. Of course, not all combinations are possible (e.g., if  $p_1$  is before  $p_2$  and  $p_2$  is before  $p_3$  there  $p_1$  must be before  $p_3$ ). For simplicity, we assume independency between the predicate pairs for the moment. In that case for each predicate pair 14 variations are possible. Thus, we have to calculate the power of 14 to the number of possible predicate pairs ( $n$  is the number of predicates in the pattern):

$$\begin{aligned} \#temporalRestrictions &= 14^{numPredPairs} \\ &= 14^{\frac{n \cdot (n-1)}{2}} \end{aligned}$$

Table 5 shows the number of temporal restrictions for different pattern sizes if five predicates are used.

Altogether the number of possible patterns grows very high if the number of used predicates in patterns is large. First of all, many “basic patterns” are created in this case. For each of these patterns many variable unifications are possible (depending on the number of variables in all predicates of the pattern). In the last step for each of the unified patterns many variations of temporal restrictions are possible. This shows that a modification of the algorithm has to be done in order to make it applicable to more complex situations.

## 7. Conclusion and Future Work

The approach presented here allows for learning patterns from sequential symbolic data. As the representation consists of predicates which describe relations between different objects more complex patterns can be learned compared to propositional event sequences. The unsupervised pattern learning algorithm performs a top-down search from the most general to more specific patterns. In the first step frequent basic patterns are generated (similar to standard association rule mining). In the second step variable unification leads to an interconnection of predicates and thus to more coherent patterns. In the third step the patterns are specialized by temporal constraints. Here, Allen’s interval logic is used to interrelate the predicates in the patterns temporarily. An evaluation function was introduced, which takes into account the special properties of our pattern representation. The algorithm was implemented and tested on artificial data sets.

Although the basic algorithm has been implemented already there are many possibilities for future work. Even

though the incremental generation of new patterns just from frequent patterns excludes many unimportant patterns, the complexity of the algorithm is still very high. An interesting research direction is to find out which heuristics could reduce complexity and still lead to good patterns. Another possibility is to prevent the program from the generation of certain patterns by leaving out (temporal) relations which are not possible due to some constraints. These constraints could be identified e.g., by accessing a composition table as introduced by Allen [3] as it was done by Höppner [5]. In some cases it might be also possible to reduce the number of temporal relations. If some of the 13 relations are not needed (or do not appear in the data) they can be omitted in order to reduce complexity.

The creation of temporal restrictions is quite straightforward so far as just one temporal relation is allowed between two predicates. If the semantic is changed to “allowed temporal relations”, more than one temporal relation could be allowed for each predicate pair. This would lead to a more powerful representation of the temporal restrictiveness.

In the current version of the algorithm just frequent patterns are generated. In future versions it would be interesting to create (temporal) association rules, which can be applied for anticipating (possible) future events.

In future work the pattern learning has to be evaluated in some application context with real data in order to evaluate how beneficial such learned patterns can be. This can be done in the context of intelligent agents which should use learned patterns to improve their future behavior, e.g., by avoiding unfavorable or dangerous situations or by reusing partial plans which have been computed in similar situation before to save time.

## 8. Acknowledgements

The content of this paper is a partial result of the ASKOF project which is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant HE989/6-1. We would like to thank Björn Gottfried, Martin Lorenz, Ingo J. Timm, and Tom Wetjen for interesting discussions on the learning of patterns and complexity calculations, and for their comments how to improve this paper.

## References

- [1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., May 1993.
- [2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, pages 487–499, September 1994.
- [3] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.
- [4] E. T. Bell. Exponential numbers. *The American Mathematical Monthly*, 41(7):411 – 419, August/September 1934.
- [5] F. Höppner. Learning temporal rules from state sequences. In *Proceedings of the IJCAI’01 Workshop on Learning from Temporal and Spatial Data*, pages 25–31, Seattle, USA, 2001.
- [6] K. Koperski and J. Han. Discovery of spatial association rules in geographic information databases. In *Proceedings of the 4th International Symposium on Advances in Spatial Databases, SSD*, pages 47–66, Portland, Maine, 1995.
- [7] D. Malerba and F. A. Lisi. An ILP method for spatial association rule mining. In *Working notes of the First Workshop on Multi-Relational Data Mining*, pages 18–29, Freiburg, Germany, 2001.
- [8] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.
- [9] J. Mennis and J. W. Liu. Mining association rules in spatio-temporal data. In *Proceedings of the 7th International Conference on GeoComputation*, University of Southampton, UK, 8 - 10 September 2003.
- [10] A. Miene, A. D. Lattner, U. Visser, and O. Herzog. Dynamic-preserving qualitative motion description for intelligent vehicles. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV ’04)*, pages 642–646, June 14-17 2004.
- [11] A. Miene, U. Visser, and O. Herzog. Recognition and prediction of motion situations based on a qualitative motion description. In D. Polani, B. Browning, A. Bonarini, and K. Yoshida, editors, *RoboCup 2003 – Proceedings of the International Symposium, July 10.-11., 2003, Padua, Italy*, 2003.
- [12] J. Pitman. Some probabilistic aspects of set partitions. *The American Mathematical Monthly*, 104(3):201–209, March 1997.
- [13] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the Eight International Conference on Knowledge Discovery and Data Mining (KDD’02)*, pages 32–41, 2002.
- [14] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 283–296, 1997.

# $TSET^{MAX}$ : An Algorithm for Mining Frequent Maximal Temporal Patterns

Francisco Guil, Alfonso Bosch  
Dept. Lenguajes y Computación  
Universidad de Almería  
{fguil, abosch}@ual.es

Roque Marín  
Dept. Ing. de la Inf. y las Comunicaciones  
Universidad de Murcia  
roque@dif.um.es

## Abstract

*The incorporation of temporal semantics into the traditional data mining techniques has caused the creation of a new area called Temporal Data Mining. This incorporation is especially necessary if we want to extract useful knowledge from dynamic domains, which are time-varying in nature. However, in a lot of cases is practically a computationally intractable problem and therefore it poses more challenges on efficient processing than non-temporal techniques. Based in the inter-transactional framework, in [13] we proposed an algorithm named  $TSET$  for mining temporal patterns (sequences) from datasets. One of the main drawbacks of this algorithm is the fact that it is an Apriori-style and therefore, for each frequent pattern, all subsets of it need to be generated. In datasets with long patterns, this could degenerate into a computationally unfeasible problem. To address this problem, in this paper we present a new algorithm named  $TSET^{MAX}$  for mining frequent maximal temporal patterns. This algorithm extract the patterns using a lookahead technique and a depth first search on a temporal set-enumeration tree.*

## 1. Introduction

Data mining is an essential step in the process of knowledge discovery in databases that consists of applying data analysis and discovery algorithms that produce a particular enumeration of structures over the data [11]. There are two types of structures: models and patterns. So, we can talk about local and global methods in data mining [20]. In the case of local methods, the simplest case of pattern discovery is finding *association rules* [4]. The initial motivation for association rules was to aid in the analysis of large transactional databases. The discovery of association rules can potentially aid decision making within organizations. Another approach is integrating the data mining process into the development of Knowledge Based Systems [22].

Since the problem of mining association rules was in-

troduced by *Agrawal* in [4], a large amount of work has been done in several directions, including improvement of the *Apriori* algorithm, mining generalized, multi-level, or quantitative association rules, mining weighted association rules, fuzzy association rules mining, constraint-based rule mining, efficient long patterns mining, maintenance of the discovered association rules, etc. We want to point out the work in which a new type of association rules was introduced, the *inter-transaction association rules* [19, 18]. Temporal data mining can be viewed as an extension of this work.

Temporal data mining can be defined as the activity of looking for interesting correlations or patterns in large sets of temporal data accumulated for other purposes [9]. It has the capability of mining activity, inferring associations of contextual and temporal proximity, some of which may also indicate a cause-effect association. This important kind of knowledge can be overlooked when the temporal component is ignored or treated as a simple numeric attribute [25].

Data mining is an interdisciplinary area which has received contributions from a lot of disciplines, mainly from databases, machine learning and statistic. In [29] we found a review of three books, each one written from a different perspective. Although each perspective make strong emphasis on different aspects of data mining (efficiency, effectiveness, and validity), only when we simultaneously take these three aspects into account we may get successful data mining results. However, in the case of temporal data mining techniques, the most influential area is artificial intelligence because its work in temporal reasoning have guided the development of many of this techniques. In non-temporal data mining techniques, there are usually two different tasks, the description of the characteristics of the database (or analysis of the data) and the prediction of the evolution of the population. However, in temporal data mining this distinction is less appropriate, because the evolution of the population is already incorporated in the temporal properties of the data being analyzed.

We can found in the literature a large quantity of temporal data mining techniques. We want to highlight some

of the most representative ones. So, we can talk about sequential pattern mining [5], episodes in event sequences [21], temporal association rules mining [6, 14, 15], discovering calendar-based temporal association rules [16], patterns with multiple granularities mining [9], and cyclic association rules mining [23]. However, there is an important form of temporal associations which are useful but could not be discovered with this techniques. These are the inter-transaction associations presented in [18, 19]. The introduction of this type of associations was motivated by the observation that many real-world associations happen under certain context, such as time, place, etc. In the case of temporal context, inter-transactional associations represents associations amongst items along the dimension of time. Due to the number of potential association becomes extremely large, the mining of inter-transaction association poses more challenges on efficient processing than classical approaches. In order to make the mining of inter-transaction associations practical and computationally tractable, several methods have been proposed in [27, 26, 12].

Working in the same direction, in [13] we presented an algorithm named *TSET* based on the inter-transactional framework for mining frequent sequences (also called frequent temporal patterns or frequent temporal associations) from several kind of datasets. The improvement of the proposed solution was the use of a unique structure to store all frequent sequences. The data structure used is the well-known set-enumeration tree, commonly used in the data mining area [7, 3, 8, 10], in which the temporal semantic is incorporated.

*TSET* follows the same basic principles as most apriori-based algorithms [4]. Frequent sequence mining is an iterative process, and the focus is on a *level-wise* pattern generation. This implies that for extract a frequent sequence of length  $k$ , it must produce all  $2^k - 2$  of its subsequences since they must be frequent too (downward closure property). This exponential complexity causes the inadequacy of the method in datasets with long associations. Some of the algorithms in the literature avoid this implementing lookahead techniques, in which supersets of frequent patterns are used in order to prune off potential candidates. As we can see in [2], there are two type of methods, lattice-based methods, such as MaxEclat or Pincer-Search [28, 17], and tree-based methods, such as MaxMiner or DepthProject [7, 1]. However, all this algorithm are applicable to datasets belonging to non-temporal domains.

The aim of this paper is to propose a new algorithm named *TSET<sup>MAX</sup>* for mining frequent maximal sequences from several kind of datasets. Just like *TSET*, it uses a unique tree-based structure, the temporal set-enumeration tree. However, in this case, a depth first search on the tree with a lookahead technique is used. We want to point out that from the frequent maximal patterns dis-

covered we can generate its subsets and count their support by reading the database once. It is very important if the goal of the mining process is, for example, to extract all association rules.

The remainder of this paper is organized as follows. Section 2 gives a formal description of the problem of mining frequent maximal temporal patterns (maximal sequences) from datasets. Section 3 introduces the algorithm named *TSET<sup>MAX</sup>*. Section 4 presents an example to illustrate the basic ideas of the proposed algorithm. Conclusions and future works are finally drawn in Section 5.

## 2. Problem Description

In this section we describe our notation, some basic definition, and we introduce the goals of our algorithm.

**Definition 1** A dataset  $D$  is an ordered sequence of records  $D[0], D[1], \dots$ , where each  $D[i]$  can have  $col$  attributes,  $c[0], \dots, c[col-1]$ . The 0-attribute will be the dimensional attribute, the temporal data associated with the record, expressed in temporal units. The rest of attributes can be quantitative or categorical. We assume that the domain of each attribute is a finite subset of non-negative integers, and we also assume that the structure of time is discrete and linear. Due to every event registered has its absolute date identified, we represent the time for events with an absolute dating system [24]. In order to simplify the calculations, we transform the original dataset subtracting the date of each record from the date of the first record, the time origin.

With this generic definition of dataset, and with minimal modifications, the algorithm that we propose works with different types of sources, that is, relational databases, transactional databases and data streams.

**Definition 2** An event  $e$  is a 3-tuple  $(c[i], v, t)$ , where  $0 < i < col$ ,  $v \in dom\{c[i]\}$ , and  $t \in dom\{c[0]\}$ , that is,  $t \in \mathbb{N}$ . Events are "things that happen", and they usually represent the dynamic aspect of the world [24]. In our case, an event is related to the fact that a value  $v$  is assigned to a certain attribute  $c[i]$  with the occurrence time  $t$ . We will use the notation  $e.c$ ,  $e.v$ , and  $e.t$  to set and get the attribute, value, and time variables related to the event  $e$ . The set of all distinct pairs  $(c[i], v)$  can be also called event types.

**Definition 3** Given two events  $e_1$  and  $e_2$ , we define the  $\leq$  relation as follows:

1.  $e_1 = e_2$  iff  $(e_1.t = e_2.t) \wedge (e_1.c = e_2.c) \wedge (e_1.v = e_2.v)$
2.  $e_1 < e_2$  iff  $(e_1.t < e_2.t) \vee ((e_1.t = e_2.t) \wedge (e_1.c < e_2.c))$

We assume that a lexicographic ordering exists among the pairs (attribute, value) in the dataset.

**Definition 4** A sequence (or event sequence) is an ordered set of events  $S = \{e_0, e_1, \dots, e_{k-1}\}$ , where for all  $i < j$ ,  $e_i < e_j$ . Obviously,  $|S| = k$ . Note that different events with the same temporal unit can belong to the same sequence. Also, the same events with different temporal unit associated can belong to the same sequence. But nevertheless, in any sequence there will exist two or more pairs (attribute, value) associated to the same temporal unit. In other words, an attribute can not take two different values in the same instant.

**Definition 5** Let  $U_{tmin}$  be the minimal dimensional value associated to the sequence  $S$ . In other words,  $U_{tmin} = \min\{e_i.t\}$ , for  $e_i \in S$ . If  $U_{tmin} = 0$ , we say that  $S$  is a normalized sequence. Note that any non-normalized sequence can be transformed into a normalized one through a normalization function.

**Example 1** Let  $S_1 = \{(0, 0, 0), (1, 0, 0), (3, 0, 2)\}$ , and  $S_2 = \{(0, 0, 3), (1, 0, 3), (3, 0, 5)\}$  be two sequences.  $S_1$  is a normalized sequenced, since it has the minimal value equal to 0 for the temporal dimension. But  $S_2$  is not a normalized sequence, because its minimal value is not equal to zero. However, we can normalize  $S_2$  by subtracting its minimal value ( $U_{tmin} = 3$ ) from the temporal values as follows:  $S'_2 = \{(0, 0, 3 - 3), (1, 0, 3 - 3), (3, 0, 5 - 3)\}$ , resulting in the normalized sequence  $S'_2 = \{(0, 0, 0), (1, 0, 0), (3, 0, 2)\}$ .

Let  $U_{tmax}$  be the maximal dimensional value associated to the sequence  $S$ . This value indicates the maximum distance amongst the events belonging to the normalized sequence  $S$ . In other words,  $U_{tmax} = e_k.t$ , where  $|S| = k$ . From both, confidence and complexity points of view [18], this value will be always less or equal than a user-defined parameter called *maxspan*, denoted by  $w$ .

**Definition 6** The support (frequency) of a sequence is defined as:  $support(S) = \frac{|D_S|}{|D|}$ , where  $|D_S|$  denotes the number of occurrences of the sequence  $S$  in the dataset, and  $|D|$  is the number of records in the dataset  $D$ .

**Definition 7** A frequent sequence is a normalized sequence whose support is greater or equal than a user-specified threshold called minimum support *minsup*.

**Definition 8** A sequence is a frequent maximal sequence if and only if it is frequent and no proper super-sequence (superset) of it is frequent.

Given a dataset and *minsup*, the goal of maximal temporal pattern (or sequence) mining is to determine in the dataset all the frequent maximal sequences whose support are greater than or equal to *minsup*.

### 3 The $TSET^{MAX}$ Algorithm

In this section we describe the algorithm that we propose for the mining of frequent maximal sequences from a dataset.

```

algorithm  $TSET^{MAX}$ (dataset  $D$ , minsup  $m$ , maxspan  $w$ )
begin
  tree.init( $D$ ,  $m$ ,  $w$ );
  tree.getSequences( $D$ ,  $m$ ,  $w$ );
  Output(tree);
end;

```

Figure 1.  $TSET^{MAX}$  Algorithm

The first step consists in the creation and initialization of the data structure. The "init" method that outlines Figure 2 returns the frequent 1-sequences (frequent events) found in the dataset. It uses a Binary Search Tree auxiliary structure to compute effectively the support of the events presented in the dataset. The inorder traversal of this structure produces the 1-sequences lexicographically ordered. The "getSequences" method (see Figure 3) obtain iteratively the frequent k-sequences using a queue structure. After generating the candidate set, which consists in copying all the events to the right of a given event in the newNode, and pruning the infrequent sequences, the new node is pushing into the queue for the next kth-cycle.

```

procedure init(tree, dataset  $D$ , minsup  $m$ , maxspan  $w$ )
begin
  BST auxiliarTree =  $\emptyset$ ;
  foreach record  $i$  in  $D$ 
    foreach non-dimensional attribute  $j$  in record
      auxiliarTree.insert( $D[i].c[j]$ ,  $D[i].c[j].v$ ,  $D[i].c[0]$ );
  tree.insert(auxiliarTree.traverseInOrder( $m$ ));
end;

```

Figure 2. The code for the method init

We will use an example to illustrate the data structure employed by  $TSET^{MAX}$ . But, before doing it, let us introduce new definitions.

**Definition 9** An Extended Set-Enumeration Tree is an set of nodes, where each node can be:

- The empty set.
- A root node of  $n$  disjoint trees.

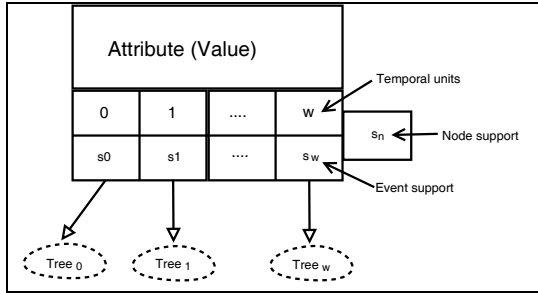
```

procedure getSequences(tree, dataset  $D$ , minsup  $m$ , maxspan  $w$ )
begin
  Queue  $Q = \emptyset$ ;
   $Q.push(tree.root)$ ;
  while ( $Q \neq \emptyset$ )
  begin
    act =  $Q.pop()$ ;
    foreach node  $n$  in act
      if  $n$  is maximal then continue;
      newNode =  $n.getCandidates()$ ;
      newNode.evaluateSupport( $D, w$ );
      newNode.pruningInFrequent( $m$ );
      if (newNode  $\neq \emptyset$ )
      begin
         $n.child = newNode$ ;
         $Q.push(newNode)$ ;
      end;
    end;
  end;
end;

```

**Figure 3. The code for the method getSequences**

**Definition 10** A node is an set of events, that is,  $node = (e_0, e_1, \dots, e_{n-1})$ , where  $e_i = (c[j], v, t, s)$ ,  $c[j]$  the attribute,  $v$  the value assigned to this attribute,  $t$  the occurrence time of the event, and  $s$  the support value associated. In Figure 4 we can see an schematic representation of a node.



**Figure 4. An schematic representation of a node**

**Definition 11** Let  $N = (e_0, e_1, \dots, e_{i-1})$  be a node of the tree. Each event  $e_i$  will have a null link or a link to a subtree rooted by the node  $N_{e_i} = (e_0, e_1, \dots, e_{j-1})$ . We will say that  $e_i$  is the parent of  $N_{e_i}$ , or simply,  $e_i = P(N_{e_i})$ . Conversely,  $N_{e_i}$  is the child of  $e_i$ , or  $N_{e_i} = C(e_i)$ . By definition, for  $e_j \in N_{e_i}$ ,  $P(e_j) = P(N_{e_i}) = N$ .

**Table 1. An example dataset**

T	$C_0$	$C_1$
0	0	0
1	1	1
2	0	0
3	1	1

This definition of ancestral relationship naturally defines the tree structure of the nodes, which is rooted at the *null* node. Our goal is to use this structure in a effective way in order to reduce the nodes generated and, consequently, the CPU time for extract frequent maximal sequences.

**Definition 12** Let  $N = (e_0, e_1, \dots, e_{j-1})$  be a node. The Head of the node  $N$  is defined as follows:  $H(N) = (e_{i_0}, e_{i_1}, \dots, e_{i_{k-1}})$ , where  $\forall e_{i_j}, 0 \leq j < k - 1, e_{i_j} = P(e_{i_{j+1}})$ . By definition, for each  $e_j \in N, H(e_j) = H(N)$ . We denote the length of  $H(N)$  as  $l$ , that is,  $l = length(H(N))$

We want to point out that for each  $e_j \in N, H(e_j) \cup e_j$  corresponds with a branch of the tree of length  $l+1$ , in other words, corresponds with a sequence of length  $l + 1$ . The support value associated with the event  $e_j$  is the support of the sequence  $H(e_j) \cup e_j$ .

**Definition 13** A node  $N = (e_0, e_1, \dots, e_{i-1})$  is a possibly maximal node if and only if for each  $e_j, 0 \leq h < i - 1, e_j < e_{j+1}$  (total order).

**Definition 14** A node  $N = (e_0, e_1, \dots, e_{i-1})$  is a maximal node if  $N$  is possibly maximal and the support of  $H(N) \cup N$  is greater than the user-defined minsup.

$H(N) \cup N$  is a maximal sequence if and only if  $N$  is a maximal node and no proper superset of it is frequent.

The lasts definitions support the basic idea of the algorithm. The algorithm combines both subset infrequency and superset frequency properties pruning strategies to remove branches from consideration [7]. If  $N$  is a maximal node, the algorithm does not decompose it into subnodes, because  $H(N) \cup N$  is a superset of all possible nodes generated from it. The deep first strategy quickly tends to find the maximal patterns first in the search process [1].

## 4. Example

We will use a synthetic dataset to illustrate the basic ideas of the algorithm that we propose in this work. Suppose that, after some preprocessing, we have obtained the dataset  $D$  shown in Table 1, where  $T$  represents the temporal attribute, and  $C_0$  and  $C_1$  two descriptive attributes. We assume that *minsup* is set at 50% and *maxspan* at 1 temporal unit.

First, we show the nodes studied by the *TSET* algorithm. As we said before, *TSET* is an *Apriory* – style algorithm and, therefore, it uses a breadth-first and level-wise strategy for extract the frequent sequences. For each node, we will show only the frequent events associated.

Let  $N_0$  be the initial node, that is, the set of frequent 1-sequences.

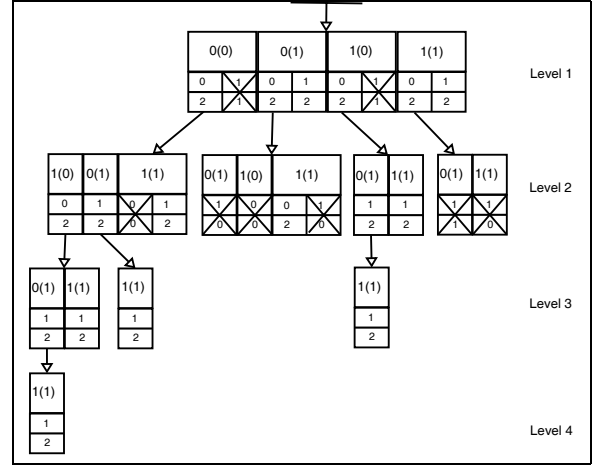
- $H(N_0) = null$   
 $N_0 = ((C_0, 0, 0, 2), (C_0, 1, 0, 2), (C_1, 0, 0, 2), (C_1, 1, 0, 2), (C_0, 1, 1, 2), (C_1, 1, 1, 2))$
- $H(N_1) = ((C_0, 0, 0, 2))$   
 $N_1 = ((C_1, 0, 0, 2), (C_0, 1, 1, 2), (C_1, 1, 1, 2))$
- $H(N_2) = ((C_0, 1, 0, 2))$   
 $N_2 = ((C_1, 1, 0, 2))$
- $H(N_3) = ((C_1, 0, 0, 2))$   
 $N_3 = ((C_0, 1, 1, 2), (C_1, 1, 1, 2))$
- $H(N_4) = ((C_1, 1, 0, 2))$   
 $N_4 = null$
- $H(N_5) = ((C_0, 0, 0, 2), (C_1, 0, 0, 2))$   
 $N_5 = ((C_0, 1, 1, 2), (C_1, 1, 1, 2))$
- $H(N_6) = ((C_0, 0, 0, 2), (C_0, 1, 1, 2))$   
 $N_6 = ((C_1, 1, 1, 2))$
- $H(N_7) = ((C_1, 0, 0, 2), (C_0, 1, 1, 2))$   
 $N_7 = ((C_1, 1, 1, 2))$
- $H(N_8) = ((C_0, 0, 0, 2), (C_1, 0, 0, 2), (C_0, 1, 1, 2))$   
 $N_8 = ((C_1, 1, 1, 2))$

Figure 5 shows the temporal set-enumeration tree generated by the *TSET* algorithm. This structure stores a total of 15 frequent sequences.

Now, we show how the *TSET*<sup>MAX</sup> algorithm works. As we said before, *TSET*<sup>MAX</sup> uses a deep-first search and a lookahead strategy for mining frequent maximal sequences.

- $H(N_0) = null$   
 $N_0 = ((C_0, 0, 0, 2), (C_0, 1, 0, 2), (C_1, 0, 0, 2), (C_1, 1, 0, 2), (C_0, 1, 1, 2), (C_1, 1, 1, 2))$

Since  $N_0$  is not a possible maximal node, the algorithm generates new candidate nodes, copying all the events to the right of a given event in the new nodes. As we want to extract normalized sequences, the 1-sequences  $(C_0, 1, 1, 2)$ , and  $(C_1, 1, 1, 2)$  will not generate new nodes.



**Figure 5. The extended set-enumeration tree structure generated by *TSET***

- $H(N_1) = ((C_0, 0, 0, 2))$   
 $N_1 = ((C_1, 0, 0, 2), (C_0, 1, 1, 2), (C_1, 1, 1, 2))$

Here, we want to highlight that the candidate node was:

$$N_{1_{old}} = ((C_1, 0, 0, 2), (C_1, 1, 0, 0), (C_0, 1, 1, 2), (C_1, 1, 1, 2))$$

which is not a possible maximal node (as we can see in the events one and two, has two different values are assigned to  $C_1$  in the same instant). However, after pruning, the node  $N_1$  becomes a possible maximal, and after the counting process, it support value is assigned to 2, that is, it is a maximal node. As  $H(N_1) \cup N_1$  does not has a frequent superset, we just found a frequent maximal sequence:

$$S_1 = ((C_0, 0, 0, 2), (C_1, 0, 0, 2), (C_0, 1, 1, 2), (C_1, 1, 1, 2)), s = 2$$

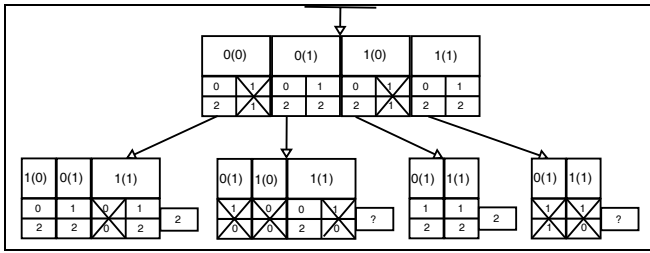
- $H(N_2) = ((C_0, 1, 0, 2)), s = 2$   
 $N_2 = ((C_1, 1, 0, 2))$

The frequent sequence  $H(N_2) \cup N_2$  has a frequent superset ( $S_1$ ). Therefore, it is discarded.

- $H(N_3) = ((C_1, 0, 0, 2))$   
 $N_3 = ((C_0, 1, 1, 2), (C_1, 1, 1, 2)), s = 2$

As in the previous case, this maximal node is discarded too.

Figure 6 shows the temporal set-enumeration tree generated by the *TSET*<sup>MAX</sup> algorithm. In this case, this structure stores a total of 3 maximal nodes, but only one frequent maximal sequence.



**Figure 6. The extended set-enumeration tree structure generated by  $TSET^{MAX}$**

## 5. Conclusions and Future Works

In this paper, we have presented  $TSET^{MAX}$ , an effective algorithm that uses a lookahead technique and deep first search on a tree-based data structure to discover frequent maximal sequences (maximal inter-transactional associations) from datasets. Although the enhancement of the method is significantly superior, in terms of computing effort, mining this type of associations is still a computationally intensive problem. Several optimizations techniques are being devised to speed up the discovery of sequences. In particular, we are working in the adaptation of the techniques proposed in the literature to reduce the number of database passes, and therefore develop an algorithm which can deal with large and dense synthetic and real datasets in an efficient way.

## References

- [1] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In *Proc. of the 6th ACM SIGKDD Int. Conf. on Knowledge discovery and Data Mining, August 20-23, Boston, MA, USA, 2000*, pages 108–118. ACM, 2000.
- [2] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent item sets. *Parallel and Distributed Computing*, 61:350–371, 2001.
- [3] C. C. Aggarwal. Towards long pattern generation in dense databases. *SIGKDD Explorations*, 3(1):20–26, 2001.
- [4] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proc. of the ACM SIGMOD Int. Conf. on Management of Data, Washington, D.C., May 26-28, 1993*, pages 207–216. ACM Press, 1993.
- [5] R. Agrawal and R. Srikant. Mining sequential patterns. In P. S. Yu and A. L. P. Chen, editors, *Proc. of the 11th Int. Conf. on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, pages 3–14. IEEE Computer Society, 1995.
- [6] J. M. Ale and G. H. Rossi. An approach to discovering temporal association rules. In *Proc. of the 2000 ACM Symposium on Applied Computing, Villa Olmo, Via Cantoni 1, 22100 Como, Italy, March 19-21, 2000*, pages 294–300. ACM, 2000.
- [7] R. J. Bayardo. Efficiently mining long patterns from databases. In L. M. Haas and A. Tiwary, editors, *Proc. of the ACM SIGMOD Int. Conf. on Management of Data (SIGMOD 1998), June 2-4, 1998, Seattle, Washington, USA*, pages 85–93. ACM Press, 1998.
- [8] F. Berzal, J. C. Cubero, N. Marín, and J. M. Serrano. TBAR: An efficient method for association rule mining in relational databases. *Data & Knowledge Engineering*, 37:47–64, 2001.
- [9] C. Bettini, X. S. Wang, and S. Jajodia. Testing complex temporal relationships involving multiple granularities and its application to data mining. In *Proc. of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, 1996, Montreal, Canada*, pages 68–78. ACM Press, 1996.
- [10] F. Coenen, G. Goulbourne, and P. Leng. Tree structures for mining association rules. *Data Mining and Knowledge Discovery*, 8:25–51, 2004.
- [11] U. Fayyad, G. Piatetky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AIMagazine*, 17(3):37–54, 1996.
- [12] L. Feng, J. X. Yu, H. Lu, and J. Han. A template model for multidimensional inter-transactional association rules. *The VLDB Journal*, 11:153–175, 2002.
- [13] F. Guil, A. Bosch, and R. Marín. TSET: An algorithm for mining frequent temporal patterns. In *Proc. of the First Int. Workshop on Knowledge Discovery in Data Streams, in conjunction with the ECML/PKDD 2004 Conference*, pages 65–74, 2004.
- [14] C. H. Lee, C. R. Lin, and M. S. Chen. On mining general temporal association rules in a publication database. In N. Cercone, T. Y. Lin, and X. Wu, editors, *Proc. of the 2001 IEEE Int. Conf. on Data Mining, 29 November - 2 December 2001, San Jose, California, USA*, pages 337–344. IEEE Computer Society, 2001.
- [15] J. W. Lee, Y. J. Lee, H. K. Kim, B. H. Hwang, and K. H. Ryu. Discovering temporal relation rules mining from interval data. In *Proc. of the 1st EurAsian Conf. on Information and Communication Technology (Eurasia-ICT 2002), Shiraz, Iran, October 29-31, 2002*, volume 2510 of *Lecture Notes in Computer Science*, pages 57–66. Springer, 2002.
- [16] Y. Li, P. Ning, X. S. Wang, and S. Jajodia. Discovering calendar-based temporal association rules. *Data & Knowledge Engineering*, 44:193–218, 2003.
- [17] D. Lin and Z. M. Kedem. Pincer-search: An efficient algorithm for discovering the maximum frequent set. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):553–566, 2002.
- [18] H. Lu, L. Feng, and J. Han. Beyond intra-transaction association analysis: Mining multi-dimensional inter-transaction association rules. *ACM Transactions on Information Systems (TOIS)*, 18(4):423–454, 2000.
- [19] H. Lu, J. Han, and L. Feng. Stock movement and n-dimensional inter-transaction association rules. In *Proc. of the Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'98), Seattle, Washington, June 1998*, pages 12:1–12:7, 1998.
- [20] H. Mannila. Local and global methods in data mining: Basic techniques and open problems. In P. Widmayer, F. Triguero,

- R. Morales, M. Hennessey, S. Eidenbenz, and R. Conejo, editors, *In Proc. of the 29th Int. Colloquium on Automata, Languages and Programming (ICALP 2002), Malaga, Spain, July 8-13, 2002*, volume 2380 of *Lecture Notes in Computer Science*, pages 57–68. Springer, 2002.
- [21] H. Mannila, H. Toivonen, , and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.
- [22] C. Ordonez, C. A. Santana, and L. de Braal. Discovering interesting association rules in medical data. In D. Gunopulos and R. Rastogi, editors, *Proc. of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Dallas, Texas, USA, May 14, 2000*, pages 78–85, 2000.
- [23] B. Özden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proc. of the 14th Int. Conf. on Data Engineering, February 23-27, 1998, Orlando, Florida, USA*, pages 412–421. IEEE Computer Society, 1998.
- [24] A. K. Pani. Temporal representation and reasoning in artificial intelligence: A review. *Mathematical and Computer Modelling*, 34:55–80, 2001.
- [25] J. F. Roddick and M. Spiliopoulou. A survey of temporal knowledge discovery paradigms and methods. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):750–767, 2002.
- [26] A. K. H. Tung, H. Lu, J. Han, and L. Feng. Breaking the barrier of transactions: Mining inter-transaction association rules. In *Proc. of the 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, August 15-18, 1999, San Diego, CA, USA*, pages 297–301. ACM Press, 1999.
- [27] A. K. H. Tung, H. Lu, J. Han, and L. Feng. Efficient mining of intertransaction association rules. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):43–56, 2003.
- [28] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In D. H. H. Mannila and D. Pregibon, editors, *Proc. of the 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD-97), Newport Beach, California, USA, August 14-17, 1997*, pages 283–286. AAAI Press, 1997.
- [29] Z. H. Zhou. Three perspectives of data mining (book review). *Artificial Intelligence*, 143:139–146, 2003.

# Real-Time Monitoring and Alerting of Web Usage

Andrew Morrison  
Department of Computer Science, RPI  
Troy, NY 12180

John Punin  
Oracle Corporation  
Redwood Shores, CA.

Mukkai S. Krishnamoorthy  
Department of Computer Science, RPI  
Troy, NY 12180

## Abstract

*Web usage mining is an active research area for its uses in web site maintenance and for the potential economical impact. In the past, research has focused on off-line statistical analysis, learning the user behavior and on identifying most frequently visited structures. Our paper takes an ambitious approach by proposing and studying effects with an on-line monitoring mechanism of web usage. In our system named  $W_{live}^3$ , we devise efficient real-time algorithms for identifying most visited sites and site-paths in real time. We also include features such as an advance warning when there is a potential denial of service attack. The  $W_{live}^3$  system uses LOGML [3] and the graph library of the WWWPal suite [2], and lay the foundation for future developments.*

*We conclude by analyzing performance properties of our system and propose some suggestions for future work.*

Studies and research in web usage mining have addressed these scenarios in the past. Traditionally this has been accomplished by gathering data on off-line statistics, user sessions, most visited web pages, most traversed web paths etc [3], [4]. Our paper addresses a more ambitious goal: that of identifying and classifying the visited web sites, traversed web paths in real-time. At the Computer Science Department, Rensselaer Polytechnic Institute, we have built a tool  $W_{live}^3$  that addresses these issues.

$W_{live}^3$  has been developed an open source software implemented in C with both a console and GNOME interface. While monitoring real-time events of web site usage it also provides advice as well as an instant warning mechanism to web managers. These features are extremely useful especially in preventing denial of service attacks by knowing the load on web pages (explained later in the paper). Our system also lays the foundation for future developments to enhance the existing framework.

## 1 Introduction

Popularity of a web site is in most part related to the content, presentation and construct of a web site. For commercial web sites this is of prime importance as each visit indicates a potential consumer. The consumer might either buy, be tempted to buy or recommend the site to others. This factor results in a potential for increased traffic to a web site which in turn can result in advertising revenue.

In case of commercial web sites, web managers work with designers, content producers, and programmers to popularize their web sites. Web site statistics such as URL location, number of hits determine site usage as well as web site traffic. While web managers prefer to have large number of users to visit their pages, there are times when users orchestrate a flood of visits to prevent others to visit a specific web site. This is damaging as this not only paralyzes the web site but also could result in a loss of revenue stream.

## 2 Previous Work

Web usage mining is the process of studying and discovering web user behavior from web log data. Usually the web log data collection is done over a long period of time (one day, one month, one year, etc). Later, three steps, namely, preprocessing, pattern discovery and pattern analysis [5] are carried out. Preprocessing is the technique of transforming the raw data into a usable data model. The pattern discovery step uses several data mining algorithms to extract the user patterns. Finally, pattern analysis reveals useful and interesting user patterns and trends. These steps are normally executed after the web log data is collected.

$W_{live}^3$  extracts and analyses user patterns in real-time, as the web log data is collected. Several commercial applications, such as Kantara [6], Deepmetrix [7], ClickTracks [8] and CacheFlow [9], have implemented real-time features to report user sessions. Kantara monitors and tracks

the users as they access the web pages. Visitor Intelligence from DeepMetrix uses real-time processing to update the business metrics to identify business trends. The web site can be modified to improve those business metrics. ClickTracks is installed in web pages to see user patterns. Finally CacheFlow Reporter generates User, Network Traffic, Security and Top Ten summaries Reports from web log data. CacheFlow has also real-time monitoring features. Interesting work has also been done for distributed real-time web usage mining [11] for frequent behavior patterns.  $W_{live}^3$  not only reports user patterns using LOGML (standard XML application), it also visualizes user patterns (user sessions web graph).  $W_{live}^3$  is designed for research purposes and it is an open source application.

### 3 Realization of the System

$W_{live}^3$  is realized as a set of open source tools. Its core components are as follows.

- **w3ld:** The  $W_{live}^3$  daemon accepts extended log file entries as they are generated and builds the structures describing the current state of the web server.
- **w3lc:** The  $W_{live}^3$  client is a console program for communicating with w3ld.
- **w3live:** w3live is a GNOME graphical interface providing easier access to the core features of  $W_{live}^3$  as well as providing visualization of web user graphs.

Figure 1 describes the system architecture.  $W_{live}^3$  reads web server logs in the extended log file format [1].

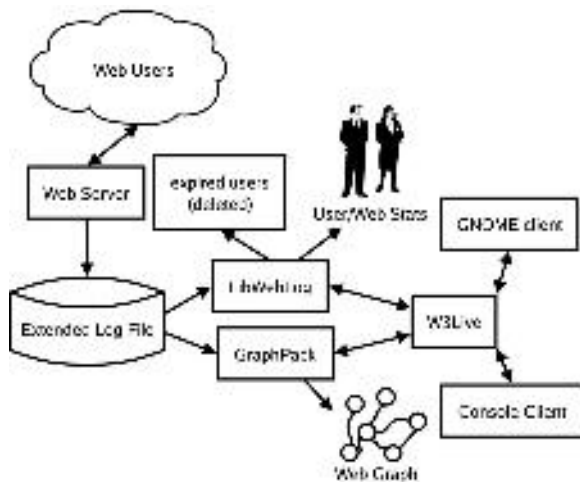


Figure 1.  $W_{live}^3$  Architecture

Building on previous work, GraphPack [10] and WWW-Pal are employed for the collection and analysis of log data. Figure 2 shows the realized  $W_{live}^3$  user interface.

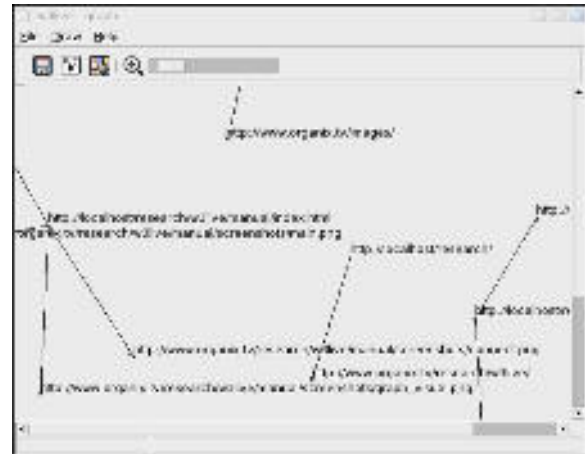


Figure 2. Graph of user clicks through website

#### 3.1 Example Analysis

The  $W_{live}^3$  software attempts to give insight into how web users are using a web site and the relationships between site structure and success in finding information.

The web site of the department of computer science at Rensselaer Polytechnic Institute serves as an example for studying real-time web usage. The site <http://www.cs.rpi.edu> serves approximately 100 requests per minute during daytime activity. It primarily serves as space for computer science faculty and students to serve class information, research and personal information.

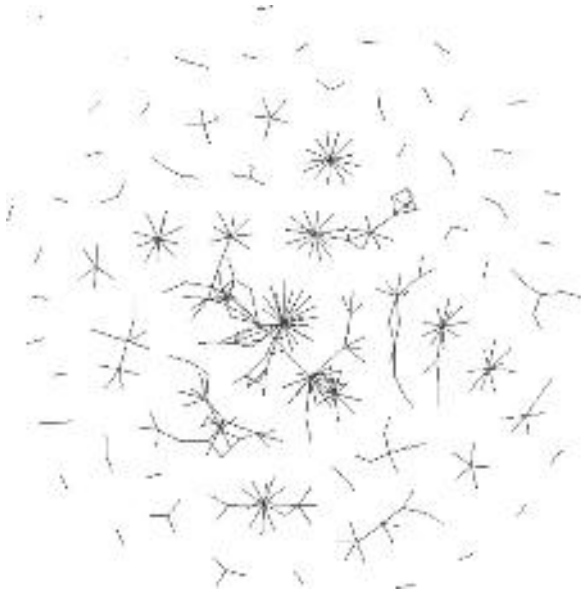


Figure 3. User Search Path

Through  $W_{live}^3$  web user graph visualization we are able to note various search patterns and judge the effectiveness of the site structure.

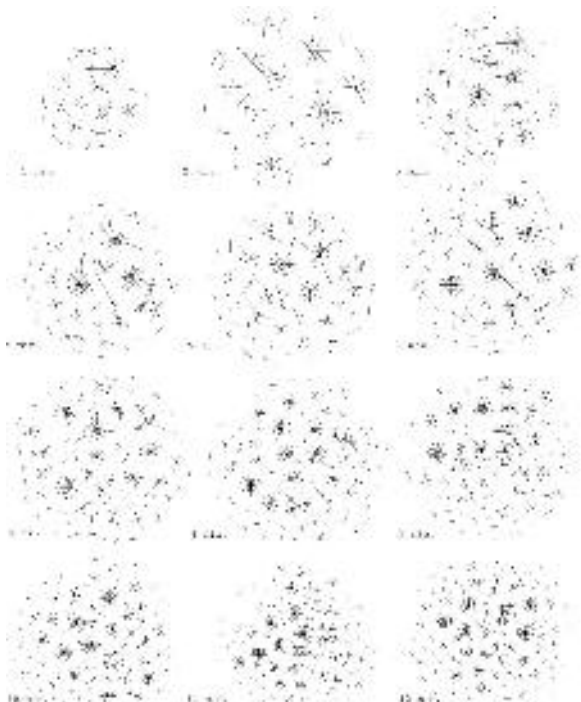
Figure 3 illustrates a user search path generated using  $W_{live}^3$ . A popular search engine has imprecisely referred the user to the site and has left the user to perform further search. A Web Manager presented with such information will be able to better mold web site structure to the user's search styles, or design better search engines based on how users are looking for information.

Activity over a 10 minute period is shown in Figure 4.



**Figure 4. Snapshot of Site Activity**

Immediate insight into the nature of user activity on the web site is given. The ad-hoc structure of the department web site is immediately clear. A Web Manager may use such information in determining the success or failure of the global web site structure.



**Figure 5. Evolution of user paths**

Figure 5 shows the evolution of user paths over time (from top left to bottom right). In this analysis of [www.cs.rpi.edu](http://www.cs.rpi.edu) over five minutes, we see several small hubs emerging and the lack of site-wide coherence becoming obvious.

## 4 Algorithms Used

The following algorithms are realized as  $W_{live}^3$ , a set of programs for real-time web usage analysis. Figure 6 gives an overview of the data acquisition loop, whereby log entries are processed into more meaningful structures. The functions **Process Log Entries** and **Terminate Expired Users** are expected to run concurrently.

Each iteration begins by updating a graph of nodes representing requested document and edges representing the relationship between a referring and target document. Building on previous work we employ the GraphPack library [2] for storage and manipulation of graphical data.

Following the graph update we update site wide statistics and user statistics. We employ WWWPal's WebLog library for this task. It is important to note that the WebLog library does much more than is needed for our task, and thus we get a slight performance degradation in these superfluous computations. Section 5 further describes such considerations.

In the following sections we describe algorithms for discovering unusual or perhaps harmful site activity.

### 4.1 Explosive Popularity

Whereby a denial of service attack may be confused with legitimate usage, it is worthwhile to differentiate the good from the bad traffic. A sudden explosion of popularity is likely due to an unexpected link from a site with more traffic. We would therefore like to determine if a single document is receiving an abnormal amount of traffic. Sudden explosive popularity may overload a web server and cause it to stop responding. Early detection would allow some time for a Web Manager to reroute requests to another server, or allow for a scaled down version of a document to be served.

We assume document popularity is governed by a Zipf distribution [12]. Each document requested is assigned a value representing its load.

Let  $d$  be the current document and  $t$  be the request count. The function  $ltime(d)$  indicates the last time document  $d$  was requested. The load  $\ell$  of document  $d$  at iteration  $t$  is computed as follows

$$\ell_t(d) = 1 + \frac{\ell_{t-1}(d)}{(now - ltime(d)) + 1} \quad (1)$$

In order to see if  $\ell_t(d)$  represents normal traffic we would like to compare it against a Zipf (power law) distribution of load averages site wide. To do this we measure the distance

```

Process Log Entries
while(entry = Read Next Log Entry)
  entry data = extract log entry data(entry)
  Lock Data(mutex)

  update web graph(entry data)
  {
    referrer = find node(entry data.referrer)
    if(referrer not in graph)
      referrer = create node(
        entry data.referrerURL)

    target = find node(entry data.target)
    if(target not in graph)
      target = create node(entry data.target)

    edge = find edge(referrer,target)
    if(edge not found)
      edge = create edge(referrer,target)

    update node(target,entry data)
    update load average(target)

    update edge(edge,entry data)
    update graph load average(
      target.loadAverage)
  }
  O(1)

  update user stats(entry data)
  {
    user = find user(entry data.address)
    if(user not found)
      user = new user(entry data)

    update user stats(user,entry data)
    update user load average(user,
      entry data)
    update site load average(
      user.loadAverage)
  }
  O(1)

  Unlock Data(mutex)

  test for events()
  {
    if((graph.loadAvg -
      threshold · graph.Std Dev) -
      target.loadAvg < 0)
      ALERT!(user is hogging bandwidth)

    if((site.request rate -
      threshold · site.Std Dev) -
      request rate < 0)
      ALERT!(current target is receiving
        abnormally heavily load)
  }
  O(1)

Terminate Expired Users
while(1)
  wait USER TIMEOUT seconds
  Lock Data(mutex)
  for each user in the list of users
    if(last access time(user) >
      USER TIMEOUT)
      terminate(user)
  Unlock Data(mutex)
  O(|users|)

```

**Figure 6. Statistics Updating Loop with Time Complexity Overview**

from the current load average,  $\ell(d)$ , to the greatest load average,  $\ell(d_{\max})$ , and compare it to an average load average, represented by the logarithmic difference between the max load average  $\ell(d_{\max})$  and the min load average  $\ell(d_{\min})$ .

$$\log \frac{\ell(d)}{\ell(d_{\max})} > \log \frac{\ell(d_{\max})}{\ell(d_{\min})} \times \frac{1}{\gamma} \quad (2)$$

Given a threshold  $\gamma$ , we may say  $\ell_t(d)$  represents explosive popularity if  $\ell_t(d)$  is more than  $\frac{1}{\gamma}$  times the slope of the power law curve further from the largest ‘normal usage’ load average

In the next section we find that explosive popularity and denial of service attacks may be confused. One should be given explicit precedence over the other so that a triggered alarm will be mutually exclusive among the two.

## 4.2 Denial of Service Monitoring

An average denial of service attack on a web server will be detected by a sudden overpowering flood of requests. Commonly these requests come from an array of machines running synchronized programs whose job is simply to make requests.

We would like to examine the temporal spacing between requests. We assume a Zipf distribution and look for spots when the space between requests is abnormally small.

Let  $c_t$  represent the time of a request at iteration  $t$ . The minimum length of time between perceived requests depends on the maximum between how quickly a web server can generate log file entries, and how quickly a single entry can be processed. The representation of  $c_t$  should reflect this minimum time in accuracy. It is likely that seconds will be too large, and milliseconds will be sufficient.

In order to tell if an unusual request rate is occurring we may look at perturbation in the log-log slope of occurrences of times between requests. It will be sufficient to count only the occurrences of the minimum and maximum of times between requests.

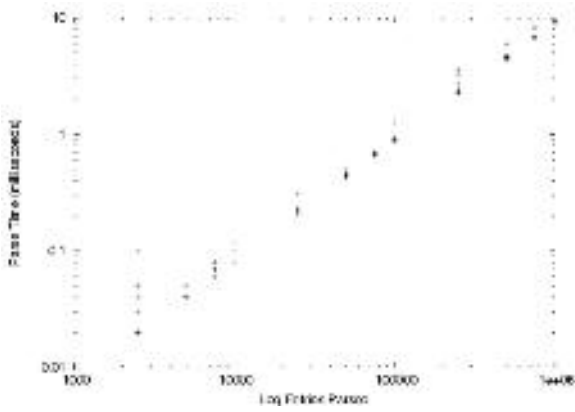
For  $\delta_t = c_t - c_{t-1}$ , let  $\delta_{\min}$  be the minimum  $\delta_t$ , and  $\delta_{\max}$  be the maximum  $\delta_t$ . Given  $\mathcal{O}(\delta_t)$ , the occurrence count for  $\delta_t$  we look for instances where

$$\log \frac{\mathcal{O}(\delta_{\max})}{\mathcal{O}(\delta_{\min})} < \log \frac{\delta_{\max}}{\delta_{\min}} \times \frac{1}{\gamma} \quad (3)$$

In situations where Equation 3 are true for a given  $\gamma$ , and Equation 2 are false, it may be said that a denial of service attack is in progress, and appropriate action may be taken.

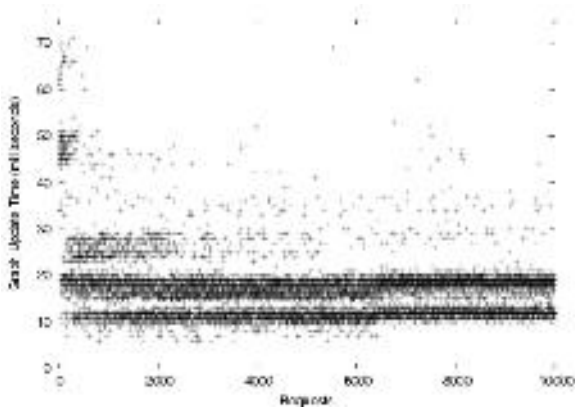
## 5 Performance of the System

The  $W_{\text{live}}^3$  software is tested for performance and comparison against the stated algorithms. For ease of implementation the stated algorithms are not followed explicitly. We show divergence between possible time complexity and real testing and explain how the system may be tuned.



**Figure 7. Log Parsing given as time to complete  $n$  log lines on a logarithmic scale.**

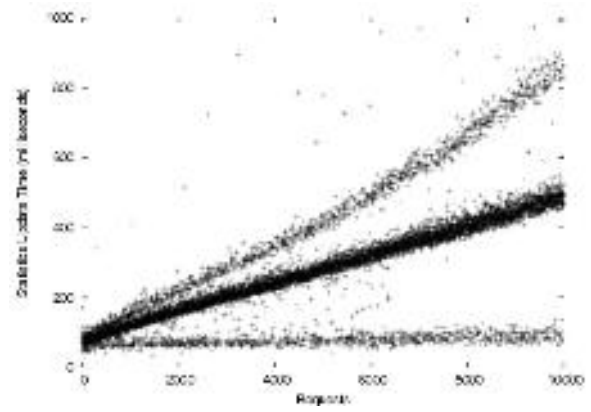
As log entries are read, we must first parse the data back out of string format. Figure 7 shows the time required to compute  $n$  entries on a logarithmic scale. Log processing on a whole has linear complexity while the processing of a single entry is  $O(1)$ .



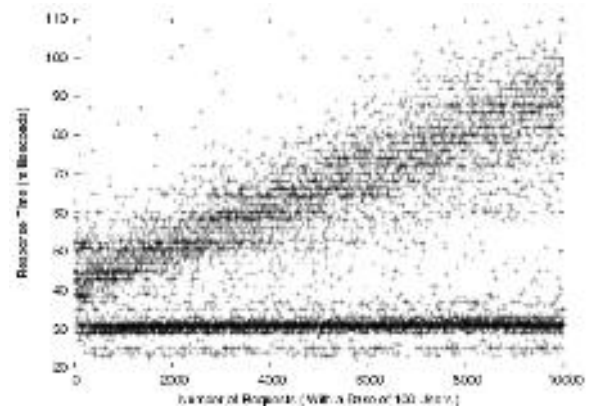
**Figure 8. Time in milliseconds for web graph update for each request**

Figure 8 shows the time in milliseconds to update the web graph at each iteration. The stratification in Figure 8 is attributed to the varying cases whereby any combination of adding a new target node, adding a new referrer node and adding a new edge may occur. Further, the initial clustering of data points between 40 and 50 milliseconds is attributed to the necessity of each request resulting in the addition of new nodes and edges.

Figure 9 shows the time in milliseconds to update the site and user statistics. It is important to note that more



**Figure 9. Statistic Update Times With Constantly Increasing Number of Users**



**Figure 10. Statistic Update Times With a Constant 100 Users**

and more users and documents are being added as more and more requests are being made and thus the complexities being represented are a function of request count, user list count and document list count.

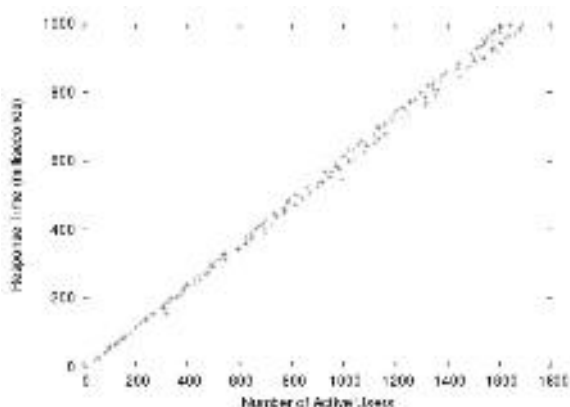
The linear complexity represents the vast majority of requests. The linearity is produced by scans over the list of users, both as searches and scanning for users who are likely to have finished their sessions.

The  $O(1)$  complexity represents file transfer errors, user requests not of type GET or unrecoverable log entry anomalies. In each of these cases, minimal statistical updating is performed.

The higher stratum represents a worst case of  $O(users^2)$  complexity occurring when a user's session is terminated. The list of users is scanned on occasion for user's whose sessions have expired. If a session is found

to be stale, they are removed from a list, requiring another scan through the list of users. This is the price paid for building on top of an off-line mining package and falls into the class of needed improvements.

Figure 10 represents the time complexity for updating statistics with a constant base of 100 users. In this case the linear time complexity is attributed to the growing number of documents being tracked as the web graph. The relatively slow growth is not noticeable in Figure 9 as it is dominated by the increasing difficulty of user management as more and more users are added.



**Figure 11. Times for Building the List of Active Users**

When studying web usage, perhaps the most common task is to request a list of currently active users. Figure 11 shows the number of active users on a server versus the number of milliseconds needed to compile the list of users and respond. We see that compiling the list is  $O(|users|)$  as expected for such a task.

## 6 Conclusion

This paper presents a system for efficient real-time monitoring of web sites. The advantages of this system are:

- It is an open source system.
- It is easily extendible to include monitoring of new events.
- It provides a real-time visualization of user behavior.
- It provides advance warnings of denial of service attacks.

Future research will focus on further improving the efficiency of weblog library. We are also planning on enhancing  $W_{live}^3$  system to capture the semantics of the visited structures in real-time. Semantic information such as

common keywords, geographic location and external referers will help to offer more accurate advance warnings and improve visualization. It will also lay the foundation for future semantic web mining research.

## References

- [1] Phillip M. Hallam-Baker, Brian Behlendorf. *Extended Log File Format W3C Working Draft WD-logfile-960323* <http://www.w3.org/TR/WD-logfile.html>
- [2] Punin, J. R., *WWWPal Suite for Analysis and Organization of Web Sites*, Ph. D. Thesis, August 2003, RPI, Troy, NY.
- [3] Punin, J.R., M. Krishnamoorthy and M. Zaki, LOGML - Log Markup Language for Web Usage Mining, in *WebKDD Workshop of SIGKDD*, 2001.
- [4] Cooley, R., B. Mobasher and J. Srivastava, Web Mining: Information and Pattern Discovery of the World Wide Web, in *Proceedings of the 9th IEEE International Conference on Tools with AI*, November 1997.
- [5] Srivastava, J., R. Cooley, M. Deshpande, P-T. Tan. Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data, in *Proceedings of SIGKDD Explorations*, 2000.
- [6] Kantara <http://www.kantara.co.uk/>
- [7] DeepMetrix Mining - Visitor Intelligence Service <http://www.deepmetrix.com/>
- [8] ClickTracks - <http://www.clicktracks.com/>
- [9] CacheFlow Reporter - <http://www.cacheflow.com/>
- [10] Krishnamoorthy, M.S., F. Oxaal, U. Dogrusoz, D. Pape A. Robayo, R. Koyanagi, Y. Hsu, D. Hollinger and A. Hashmi. GraphPack: Design and Features. *Software Visualization*, vol. 7, pages 83–100, 1996.
- [11] Florent Masseglia, Maguelonne Teisseire, Pascal Poncellet. Real Time Web Usage Mining: A Heuristic Based Distributed Miner, in *Proceedings of the RIDE 2002*, 2002.
- [12] Steven Glassman, A caching relay for the World Wide Web, *Computer Networks and ISDN Systems*, vol. 27, pages 165–173, 1994.