3rd International Workshop on Mining Temporal and Sequential Data (TDM-04)

Workshop Chairs:
K. P. Unnikrishnan
Ramasamy Uthurusamy
Jiawei Han

August 22, 2004
Seattle, USA

Table of Contents

Foreword
K. P. Unnikrishnan, Ramasamy Uthurusamy, and Jiawei Han i

Data Mining for Financial Time Series: Learning to Trade via Direct Reinforcement
John Moody (Invited Talk) 1

Research Papers

Predicting Land Climate Using Ocean Data
S. Boriah, G. Simon, N. Maniratan, M. Steinbach, V. Kumar, S. Klooster, and C. Potter 2

Density-Based Clustering of Time-Series Subsequences
Anne Denton 14

Clustered Segmentations
Aristides Gionis, Heikki Mannila and Evimaria Terzi 22

Fast Algorithms for Frequent Episode Discovery in Event Sequences
Srivatsan Laxman, P.S. Sastry and K. P. Unnikrishnan 33

Extracting Time-Ordered pairs of Similar Subsequences by Time-Warping Approach
Sohhei Morita, Koich Furukawa, and Tomonobu Ozaki 41

Everything you know about Dynamic Time Warping is Wrong
Chotirat Ann Ratanamahatana and Eamonn Keogh 50

FastTDW: Toward Accurate Dynamic Time Warping in Linear time and Space
Stan Salvador and Philip Chan 61

Finding Reference Affinity Groups in Trace Using Sampling Method
Chengliang Zhang, Yutao Zhong, Chen Ding, and Mitsunori Ogihara 72

Posters

Frequent Pattern Mining in Real-Time - First Results
Rajanish Dass and Ambuj Mahanti 84

A Specialized Binary Tree for Financial Time Series Representation
Tak-Chung Fu, Fe-Lai Chung, Robert Luk, and Chak-Man Ng 96

Mining temporal Sequences to Discover Interesting Pattern
Edwin Heierman, III, G. Michael Youngblood, and Diane J. Cook 104

An Efficient Method for Finding Emerging Large Itemsets
Susan Imberman, Abdullah Uz Tansel, and Eric Pacuit 112

Order and Mess in Text Categorization: Why Using Sequential Patterns to Classify
S. Jaillet, A. Laurent, Maguelonne Teisseire, and J. Chauche 122

Finding Complex Patterns over Data Streams
Leila Kaghazian, Dennis McLeod, and Resa Sadri 130

Default Prediction from the Bond Market
Abhinanda Sarkar and Debasis Bal 138

Foreword

This is the third Temporal Data Mining (TDM) Workshop at KDD. This year's title "Workshop on Mining Temporal and Sequential Data" reflects our initiative to broaden the scope.

Though much of the data in large databases either have explicit or implicit timing/ordering information, many of the successful KDD applications still involve finding static relationships from these databases. Examples of temporal / sequential data include financial data, data from manufacturing plants, data obtained from scientific and engineering processes, sensor data, stream data, weather and geophysical data, traffic-flow data from physical and electronic networks, multi-media and speech data, genomic and other bio-informatics data, brain imaging data, etc. Many of the systems we treat as static, whether the world-wide-web or the human brain, are truly dynamic. Mining of data from these systems would be much easier if the temporal / dynamic nature of these systems is directly addressed.

The aim of this workshop is to bring out the above and address challenging technical issues to achieve it. A large, inter-disciplinary effort would be needed for this. The notes of the workshop contain research papers selected for presentation over four sessions: some as talks, and some as posters. In addition, the workshop includes an invited presentation in the morning and a panel / group discussion at the end. Notes from the panel discussion titled "Where should TDM go?" would be posted (along with all the research papers to be presented in the workshop) at the workshop web site.

We thank all the authors who submitted papers and all those who helped with the review process. We look forward to a very lively and fun workshop at KDD-2004 in Seattle.

Some of you may be curious about the back cover. Though it may look as child's play, a lot of information about the content and production of this volume is contained on the back cover. Details are available at the workshop website.

We would like to thank General Motors Corporation for help with printing of the workshop notes.

K. P. Unnikrishnan
Ramasamy Uthurusamy
Jiawei Han

Workshop Co-Chairs

Data Mining for Financial Time Series: Learning to Trade via Direct Reinforcement

John Moody
International Computer Science Institute, Berkeley
J E Moody & Company LLC, Portland
Email: moody@icsi.berkeley.edu

Abstract:

This talk presents two themes: (1) that properties of financial price series pose unique challenges for data mining and (2) that Direct Reinforcement learning can provide decision support solutions in dynamic domains such as financial market trading.

Financial price series have a number of characteristics that distinguish them from other domains for temporal data mining. The efficient market hypothesis suggests that financial price series should behave as purely random processes (martingales) with no repeated patterns or predictable behavior. However, most practitioners and academics agree that financial time series obey a number of "stylized facts". These include nonstationarity, very high noise, clustering of volatility, co-movements of similar series, heavy tails and extreme events. There is evidence to suggest that predicting direction of financial markets is actually easier than making point forecasts. Even with such structures, financial price series present unusual challenges for analysis that defy standard methods.

Most machine learning and data mining research has focused on supervised or unsupervised algorithms, which treat data generating processes as static. However, such standard non-dynamic methods can't capture temporal dependencies that arise when actions affect the state of the system. When trading financial markets, for example, transaction costs induce temporal dependencies. Reinforcement learning is a dynamic paradigm based upon maximizing utility via trial and error experience that can solve such problems. While traditional value function-based RL methods have failed to yield many significant real-world solutions, Direct Reinforcement (policy gradient) methods provide powerful and efficient new means for temporal sequence mining. I present a brief overview of DR and illustrate its application to discovering profitable trading strategies. Examples include asset allocation between stocks and bonds and trading the foreign currency markets.

Reference:

Learning to Trade via Direct Reinforcement, John Moody and Matthew Saffell, Special issue on Financial Engineering, IEEE Transactions on Neural Networks, Vol. 12, No. 4, July 2001.

Predicting Land Temperature Using Ocean Data

Shyam Boriah **György Simon** **Maniratan Naorem**
Michael Steinbach **Vipin Kumar**

Department of Computer Science and Engineering
University of Minnesota
sboriah,gsimon,naorem,steinbac,kumar@cs.umn.edu

Steven Klooster

California State University
Monterey Bay
klooster@gaia.arc.nasa.gov

Christopher Potter

NASA Ames Research Center
cpotter@mail.arc.nasa.gov

ABSTRACT

To analyze the effect of the oceans and atmosphere on land climate, Earth Scientists have developed climate indices, which are time series that summarize the behavior of selected regions of the Earth's oceans and atmosphere. In the past, Earth scientists have used observation and, more recently, eigenvalue analysis techniques, such as principal components analysis (PCA) and singular value decomposition (SVD), to discover climate indices. Recently, an alternative clustering-based methodology has been developed for identifying climate indices. This paper presents preliminary work evaluating the effectiveness of Sea Surface Temperature (SST) and Sea Level Pressure (SLP) cluster-based indices in predicting land temperature and their relative performance with respect to known climate indices. As part of our effort, we studied the North Atlantic Oscillation (NAO) index, which is known to impact land temperature in the US, and its cluster-based counterpart, which is derived using daily SLP data from the Atlantic Ocean for a 25 year period (1979-2003). We also studied the predictive power of 28 SST clusters that were identified as the most promising clusters derived from monthly SST data for a 41-year period (1958-1998) [14]. These clusters were shown to be similar to well known climate indices in terms of area weighted correlation to global land temperature, and were considered as prime candidates for further evaluation. Our preliminary results are very encouraging. They show that many of the cluster-based indices can outperform known climate indices in predicting anomalies in land temperature for certain parts of the world.

KDD 2004 Seattle, WA, USA

Categories and Subject Descriptors

I.5.3 [**Pattern Recognition**]: Clustering; I.5.4 [**Pattern Recognition**]: Applications—*Climate*

Keywords

clustering, time series, climate index, Earth science data, mining scientific data

1. INTRODUCTION

It is well known that ocean, atmosphere and land processes are highly coupled, i.e., climate phenomena occurring in one location can affect the climate at a far away location. Indeed, understanding these climate teleconnections is critical for finding the answer to questions such as how the Earth's climate is changing and how ecosystems respond to global environmental change. A common way to study such teleconnections is by using climate indices [9, 10], which distill climate variability at a regional or global scale into a single time series. For example, El Niño, the anomalous warming of the eastern tropical region of the Pacific, has been linked to climate phenomena such as droughts in Australia and heavy rainfall along the Eastern coast of South America [17]. Most commonly used climate indices are based on sea level pressure (SLP) and sea surface temperature (SST). Earth scientists have used observation and, more recently, eigenvalue analysis techniques, such as principal components analysis (PCA) and singular value decomposition (SVD), to discover climate indices.

In [14] an alternative clustering-based technique was presented for discovering climate indices. The use of clustering is driven by the intuition that a climate phenomenon is expected to involve a significant region of the ocean or atmosphere, and that we expect that such a phenomenon will be 'stronger' if it involves a region where the behavior is relatively uniform over the entire area. SNN clustering [4, 5, 6] has been shown to find such homogeneous clusters. Each of these clusters can be characterized by a centroid, i.e., the mean of all the time series describing the points (locations) in the cluster, and thus, these centroids represent potential

climate indices. It was demonstrated that the centroids of many clusters of SST and SLP, which were discovered using the SNN clustering algorithm, correspond to known climate indices; other clusters were found to be variants of known climate indices that could provide better predictive power for some land areas; and still other clusters represent potentially new Earth science phenomena.

This paper presents two different sets of experimental results evaluating the effectiveness of SST and SLP cluster-based indices in predicting land temperature, particularly in comparison to known climate indices. In the first set of experiments, we study the North Atlantic Oscillation (NAO) index, which is known to impact land temperature in the United States, and its cluster-based counterpart that is derived using daily SLP data from the Atlantic Ocean for a 25 year period (1979-2003). In the second set of experiments, we study the predictive power of 28 SST clusters that were identified as the most promising clusters derived from monthly SST data for a 41-year period (1958-1998) [14]. These clusters were shown to be similar to well known climate indices in terms of area weighted correlation to global land temperature, and were considered as prime candidates for further evaluation. This set of experiments first tries to answer two questions: In which land areas can temperature be predicted by the selected 28 SST clusters? How do predictions based on these SST clusters compare to those obtained from (a) models that use known climate indices and (b) models that use only temporal autocorrelation? Second, the experiments also investigate whether the use of SST clusters can augment prediction based on temporal autocorrelation, and if so, how the results compare to predictions obtained using temporal autocorrelation augmented with known climate indices.

Paper organization. Sections 2 and 3 provide a quick overview of the Earch science data and climate indices that we used in our experiments. Sections 4 and 5 discuss the two sets of experiments, respectively. Section 6 summarizes our work and indicates future directions.

Note: The pictures in this paper should be viewed in color. A pdf version of this paper with color figures can be found at http://www.cs.umn.edu/~kumar/papers/kdd04nasa.pdf.

2. EARTH SCIENCE DATA AND CHALLENGES

The Earth science data for our analysis consists of global snapshots of measurement values for a number of variables (e.g., temperature, pressure) collected for all land and sea surfaces (see Figure 1). For the analysis presented here, we focus on attributes measured at points (grid cells) on latitude-longitude spherical grids of different resolutions, e.g., global land temperature, which is available at a resolution of $0.5^\circ \times 0.5^\circ$, US land temperature available at a resolution of 8 km $\times$ 8 km, and SST, which is available for a $1^\circ \times 1^\circ$ grid, and SLP, which is available for a $2.5^\circ \times 2.5^\circ$ grid. Our analysis uses data available at daily as well as monthly intervals.

The spatial and temporal nature of Earth Science data poses a number of challenges. For instance, Earth Science time series data is noisy, has cycles of varying lengths and regularity, and can contain long term trends. In addition, such data displays spatial and temporal autocorrelation, i.e., measured values that are close in time and space tend to

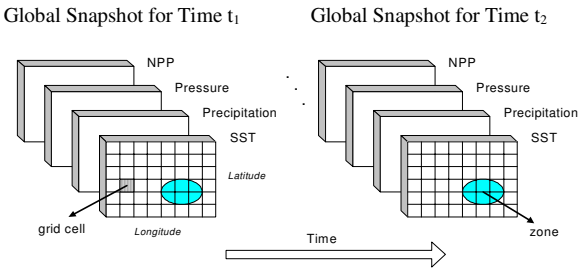


Figure 1: A simplified view of the problem domain.

be highly correlated, or similar. To handle these issues, we perform different types of preprocessing for daily and monthly data that we will describe in subsequent sections. For further details on these issues, we refer the reader to [15, 16] and [14].

3. CLIMATE INDICES

Climate indices [9, 10] are time series that capture the variability of climate for certain regions of the world. Out of the well-known indices listed in Table 1, PDO, CTI, NINO1+2, NINO3, NINO3.4, and NINO4 are based on temperature anomalies at different regions of the world, while SOI, NAO, AO, and WP are based on SLP. SOI and NAO are computed from differences in pressure between regions, while AO and PDO are obtained by PCA techniques.

Table 1: Description of well-known climate indices.

Index	Description
SOI	(Southern Oscillation Index) Measures the SLP anomalies between Darwin and Tahiti
NAO	(North Atlantic Oscillation) Normalized SLP differences between Ponta Delgada, Azores and Stykkisholmur, Iceland
AO	(Arctic Oscillation) Defined as the first principal component of SLP poleward of 20° N
PDO	(Pacific Decadel Oscillation) Derived as the leading principal component of monthly SST anomalies in the North Pacific Ocean, poleward of 20°N
QBO	(Quasi-Biennial Oscillation Index) Measures the regular variation of zonal (i.e. east-west) stratospheric winds above the equator
CTI	(Cold Tongue Index) Captures SST variations in the cold tongue region of the equatorial Pacific Ocean (6°N-6°S, 180°-90°W)
WP	(Western Pacific) Represents a low-frequency temporal function of the 'zonal dipole' SLP spatial pattern involving the Kamchatka Peninsula, southeastern Asia and far western tropical and subtropical North Pacific
NINO1+2	Sea surface temperature anomalies in the region bounded by 80°W-90°W and 0°-10°S
NINO3	Sea surface temperature anomalies in the region bounded by 90°W-150°W and 5°S-5°N
NINO3.4	Sea surface temperature anomalies in the region bounded by 120°W-170°W and 5°S-5°N
NINO4	Sea surface temperature anomalies in the region bounded by 150°W-160°W and 5°S-5°N

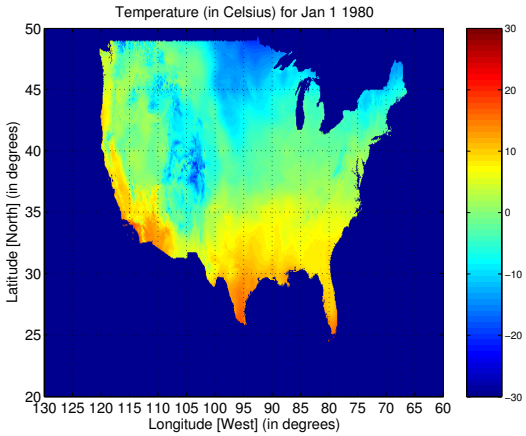


Figure 2: Average Temperature in the United States on January 1, 1980

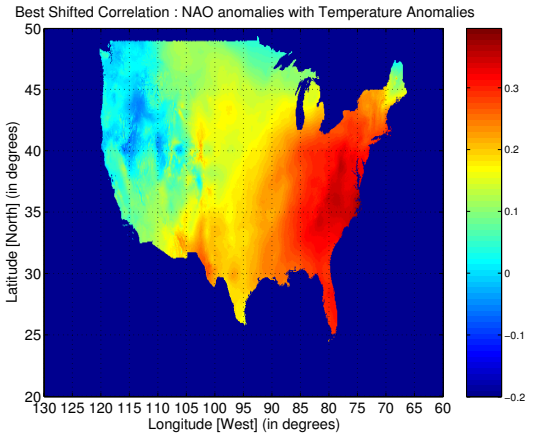


Figure 3: Best shifted correlation between NAO and temperature anomalies for 17 extended winters

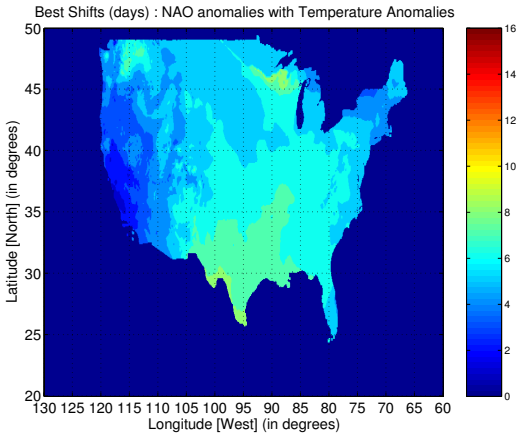


Figure 4: Shifts (days) that gave the best shifted correlation

4. COMPARISON OF NAO AND A CLUSTER-BASED VARIANT USING US LAND TEMPERATURE

NAO is one of the oldest known world weather patterns—some of the earliest descriptions of it came from seafaring Scandinavians several centuries ago. NAO refers to swings in the atmospheric sea level pressure difference between the Arctic and the subtropical Atlantic that are most noticeable during the boreal cold season (November-April) and are associated with changes in the mean wind speed and direction in the North Atlantic [8]. It is well-known in the Earth Science community that NAO influences temperature and snowfall in the Eastern United States and Europe during the cold season [7, 13].

A daily NAO index is available from 1950 to near realtime from the National Oceanic and Atmospheric Administration (NOAA) [11]. This dataset consists of NAO anomaly (departures from the mean) measurements for every day since January 1, 1950. We preprocessed this dataset by filtering out high-frequency noise using a 7-day centered running average.

We obtained daily temperature data from 1980 – 1997 from the University of Montana Numerical Terradynamic Simulation Group (NTSG) [3]. This is the highest temporal and spatial resolution dataset available for temperature in the United States. The data we have used is at a resolution of 8 km × 8 km, although the original data is available at a resolution of 1 km × 1 km. For each day, we computed the average temperature as the mean of the maximum and minimum temperature measurements, and used the average temperature to represent the temperature for that day. Average temperature is typically computed in this manner for summary-of-the-day observations [2]. For example, Figure 2 shows the average temperature (in Celsius) in the United States on January 1, 1980. To filter out high-frequency noise from the data, we use a 7-day centered running mean across the dataset. We then preprocessed the temperature data by transforming it from temperature measurements to temperature anomalies. We compute the temperature anomaly for a particular point on a particular day by subtracting the mean temperature for that day across the dataset from the temperature measurement on that day. Therefore, the final temperature data we use is a time series for each land location in the United States that contains daily temperature anomalies from 1980 – 1997.

A daily SLP dataset was obtained from NCEP Reanalysis 2 data provided by the NOAA-CIRES Climate Diagnostics Center, Boulder, Colorado, USA, at their Web site [12]. The resolution of this dataset is 2.5° × 2.5°. We preprocessed the 25 year (1979 – 2003) daily SLP dataset by normalizing the data. We first applied a filter using a 7-day centered running average to smooth high-frequency noise. Then, the time series for each SLP grid location was normalized by subtracting the mean pressure for that location. This makes the mean of each time series zero.

4.1 Impact of NAO on US Land Temperature

We have used two methodologies to examine the relationship between NAO and land temperature over the United

States during the cold season. We focus on the cold season because NAO exhibits most of its variability during the extended winter (December-March).

We have used correlation in our past studies with Earth Science data, and we use it in this study as our first method. Since NAO can have an impact on temperature in different places at different times, we take these lags into account. Thus, it is necessary to compute the correlation for various shifts. This involves shifting the two time series to simulate leads (lags) of up to 15 days, computing the correlation, and then taking the 'best' (highest positive or negative value) as the correlation. However, taking the 'best shifted correlation' for each land point, individually, can lead to two neighboring points having correlations corresponding to different shifts. Thus, we employed a 'smoothing' procedure which ensures that the 'best' shift at a point is as consistent as possible with respect to its neighboring points [14]. Since we focused only on extended winters, correlation had to be computed separately for each extended winter. For each land location, we use the shift that gives the best overall correlation over all the extended winters. Figure 3 shows the best shifted correlation of NAO anomalies and temperature anomalies for 17 extended winters between 1980 and 1997. Figure 4 shows the shifts that produced the best correlation at each point. The shifts required minimal 'smoothing' in this case. Figure 3 shows that NAO has its strongest relationship with temperature in the Eastern United States. This conforms to our expectation that NAO impacts temperature in the Eastern United States during the cold season (extended winter).

A second method that we used to evaluate the impact of NAO on land temperature is similar to the runs test from statistics [1], which can be used to decide if a data set is from a random process. A run is defined as a series of increasing values or a series of decreasing values. The number of increasing, or decreasing, values is the length of the run. For NAO and land temperature anomaly data we define a run as the number of days that both the anomalies have the same sign. We allow a small number of sign changes (tolerance) when counting runs since we would like to be able to observe very long runs even if there are a few days where the two anomalies do not have the same sign. For example, if a particular land location had temperature anomalies with the same sign at NAO anomalies for 80 days except for 3 days somewhere in between, we would still count this as a run of length 80. Using this methodology, we counted runs of all lengths for NAO and temperature anomalies at a tolerance level of 3. As in the case of correlation, we focus only on extended winters. Figure 5 shows the number of runs longer than twenty days weighted with the length of the run. Similarly, Figure 6 and Figure 7 show runs longer than 40 and 80 days, respectively.

Figure 5 shows that a large number of the runs of length 20 and longer are concentrated around a similar area where the NAO is known to have most impact. However, as we look at runs of longer lengths only, as in Figure 6 and 7, we see that the areas of significance shift away from the Eastern coastal areas of the United States. In Figure 8 we show the longest run that could be found at each land location across the US. This figure clearly shows that regions that have the longest runs are not where NAO is known to have most impact. The longest run was 111 days for seven contiguous land points in the Northwest US. In Figure 10,

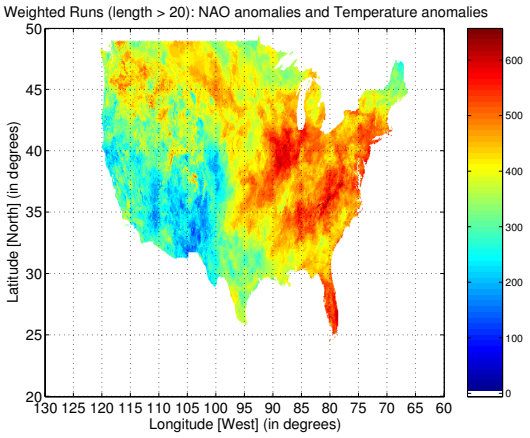


Figure 5: Weighted runs of length 20 and longer between NAO and temperature anomalies

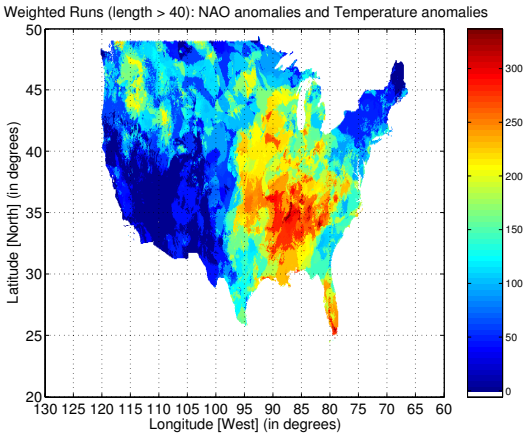


Figure 6: Weighted runs of length 40 and longer between NAO and temperature anomalies

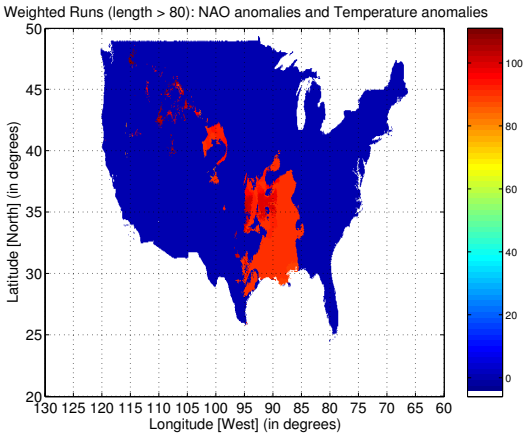


Figure 7: Weighted runs of length 80 and longer between NAO and temperature anomalies

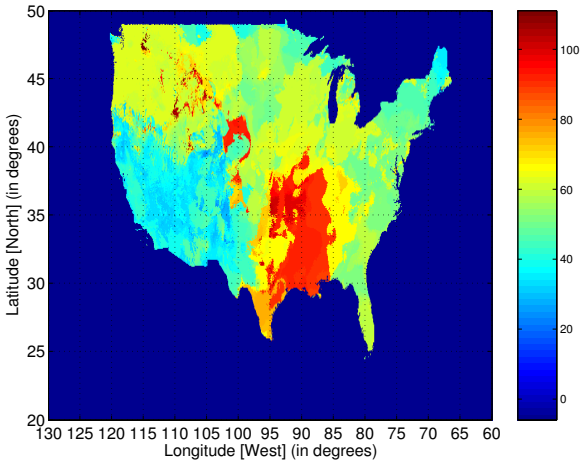


Figure 8: Longest run: NAO and Temperature anomalies

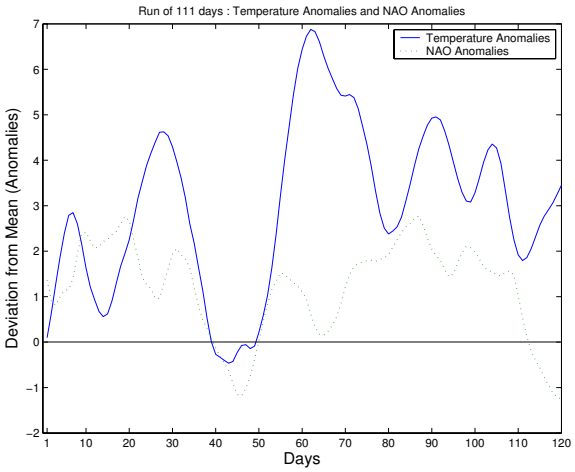


Figure 10: A 111-day long run

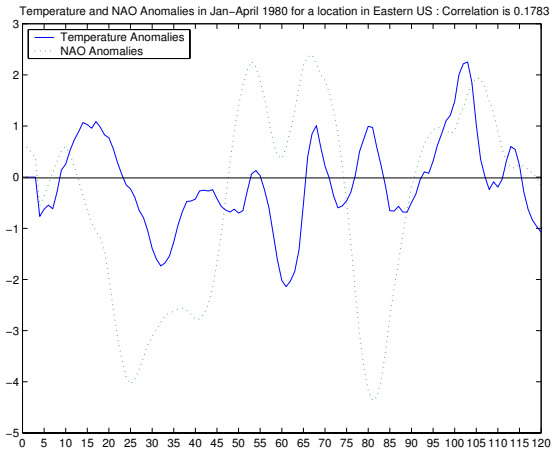


Figure 9: A single extended winter for a point in the Northeast US

we show NAO and temperature anomalies of one of these points for the extended winter (1991-1992) when this run occurred. Therefore, it appears that even though NAO is known to have most influence over land temperatures in the Eastern United States, the runs of NAO and temperature in this region are not the longest when compared to the rest of the country. This could be due to a number of reasons including the possibility that temperature near the Eastern coast is influenced much more by the ocean and is therefore prone to more fluctuation. The presence of long runs in the Southern United States indicates that NAO have influence even in areas other than the Eastern United States that are traditionally considered to be impacted by NAO. One may need to study additional factors (e.g. events occurring in the Gulf Of Mexico or Pacific Ocean) to understand when NAO does impact these areas.

A comparison of the two methodologies we have used above to evaluate the impact of NAO on land temperature shows that the two methods give us different insights into what we are trying to evaluate. Both methods have different limitations, advantages. Figure 8 would even suggest that the two methods are complementary. It is clear that correlation shows us where NAO has the most impact *consistently* over all the extended winters. These regions, as we would expect, are in the Eastern United States. A close look at NAO and temperature anomaly time series of a few points in the Eastern US showed that even though the time series visually appeared to have a clear relationship, a few days of difference in trends reduces the correlation, making the relationship appear weak. Figure 9 shows the time series for an extended winter of one such point in the Northeastern US. The NAO and temperature anomalies visually appear to have a relationship but this is not reflected in the correlation (0.18). However, when correlation was computed excluding the 45 days in the middle of the time series, the value obtained was 0.53. Therefore, correlation is susceptible to periods of difference in trend. However, even in periods of high correlation, looking at the runs, one would not see long runs since the two time series change signs in short periods of a few days. Similarly, Figure 10 shows a point that has a very long run where NAO and temperature anomalies have the same sign, but the correlation is only 0.25.

4.2 Impact of SLP Cluster-based Indices on US Land Temperature

We used the SNN clustering algorithm to generate a cluster-based variant of NAO index, and compared its performance with the standard NAO index. The SNN clustering algorithm was applied to the transformed data and 26 clusters were obtained. Figure 11 shows the clusters. The clusters are numbered in the figure for labeling purposes only.

There were 4 clusters in the Atlantic Ocean that got our attention because they were in locations close to where NAO is active. These are clusters 7, 15, 10a and 10b. We constructed indices using these clusters to compare them to the

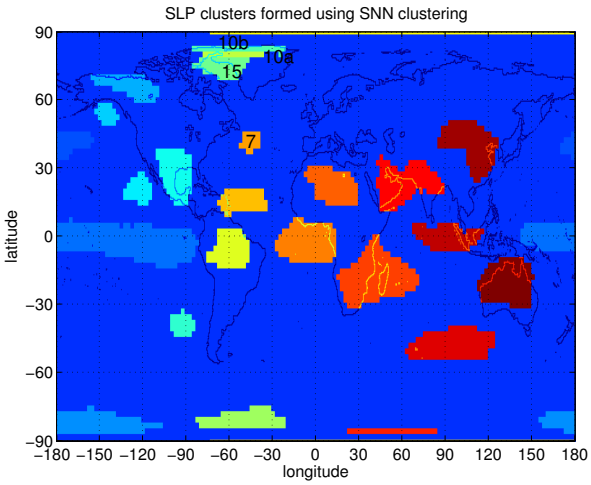


Figure 11: SLP Clusters found by SNN algorithm

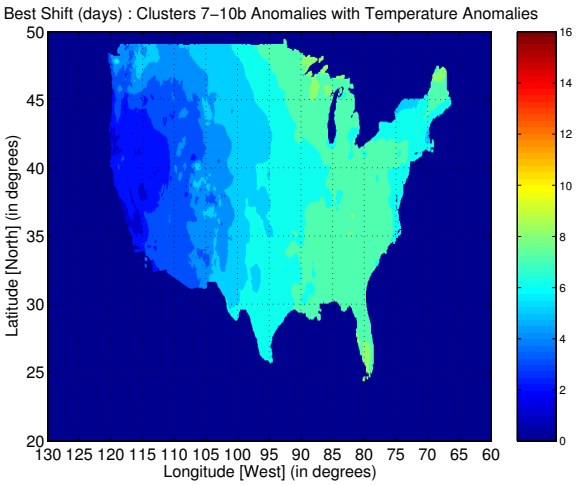


Figure 13: Shifts that gave the best shifted correlation for SLP cluster-based index

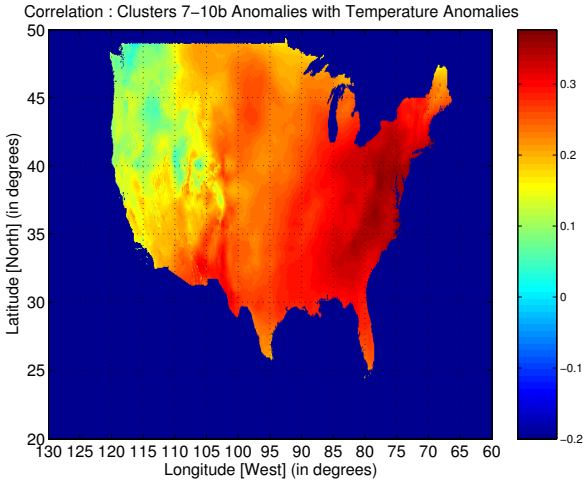


Figure 12: Best shifted correlation between SLP cluster-based index and temperature anomalies

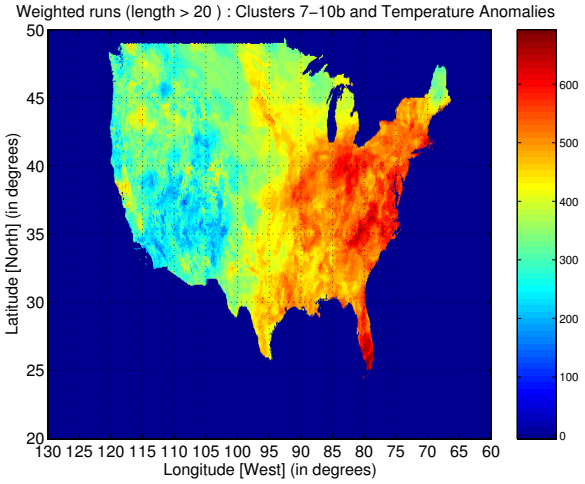


Figure 14: Weighted runs of length 20 and longer between SLP cluster-based index and temperature anomalies.

standard NAO index. We selected clusters in the North Atlantic and mid-Atlantic Ocean to generate an SLP cluster-based index. First, we took the centroid of the each of the clusters by taking the mean of the daily SLP time series of the grid locations that comprise the cluster. A 7-day centered running average filter was applied to smooth high-frequency noise. Then, the SLP time series of the centroid was transformed to an anomaly by subtracting the daily mean for each day from the measurement for that day. Then, we constructed three indices by taking the difference of the anomalies of the clusters found near Greenland (clusters 15, 10a, and 10b) and the cluster found near the Azores (cluster 7). We will present the results of only one index here (clusters 7 - cluster 10b) because it performed slightly better than the other two.

We applied the same methodology to evaluate the influence of the SLP cluster-based index on US land temperature as we used to evaluate the influence of the standard NAO index. We focus only on the extended winters, put the cluster-based index in place of the standard NAO index and perform the evaluation using an identical procedure (i.e. all the parameters such as tolerance of 3 days were the same). Figures 12 and 13 show the best shifted correlation and best shifts obtained for our cluster-based index.

We show the runs of length 20, 40 and 80 days or longer in Figures 14, 15, and 16, respectively. The runs of maximum length for each location are shown in Figure 17. We present a comparison of the performance of the standard NAO index and our SLP cluster-based index in the next subsection.

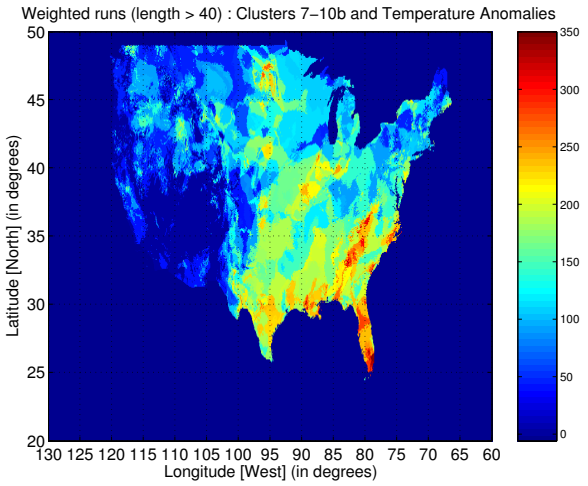


Figure 15: Weighted runs of length 40 and longer between SLP cluster-based index and temperature anomalies.

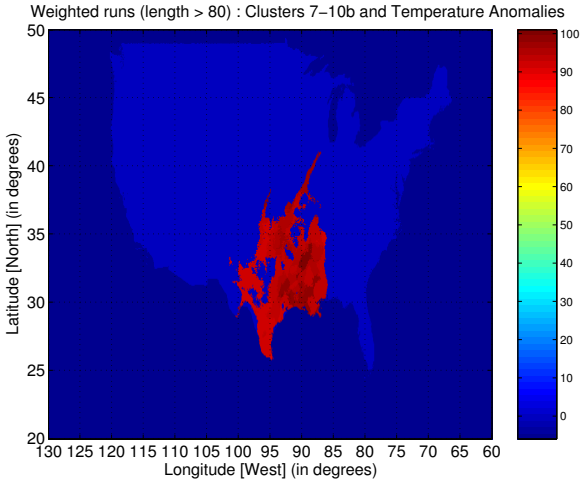


Figure 16: Weighted runs of length 80 and longer between SLP cluster-based index and temperature anomalies.

4.3 Comparison of NAO and SLP Cluster-based Index

We compared the performance of the two indices by looking at their performance in each of the experiments that were performed above. Figure 18 shows the difference between the best shifted correlation of the two indices. Positive numbers indicate that our cluster-based index performed better, while negative numbers indicate that the standard NAO index performed better. We see that our cluster-based index performs as well as the standard NAO index in the Eastern United States. There are very few regions where our index does worse, and large regions (especially near the West coast) where it does much better than the standard NAO index.

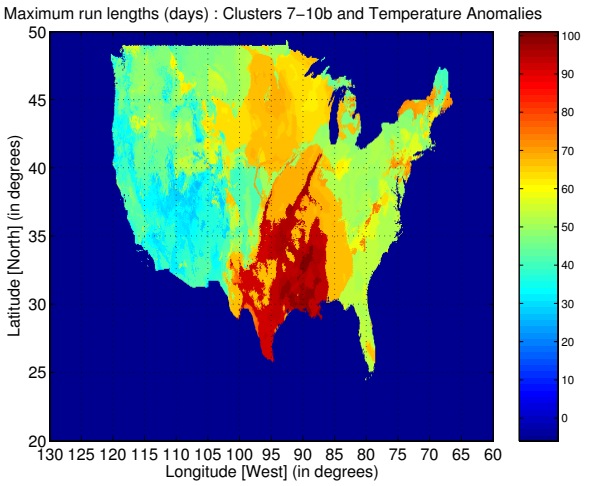


Figure 17: Longest run: Clusters 7-10b with temperature anomalies.

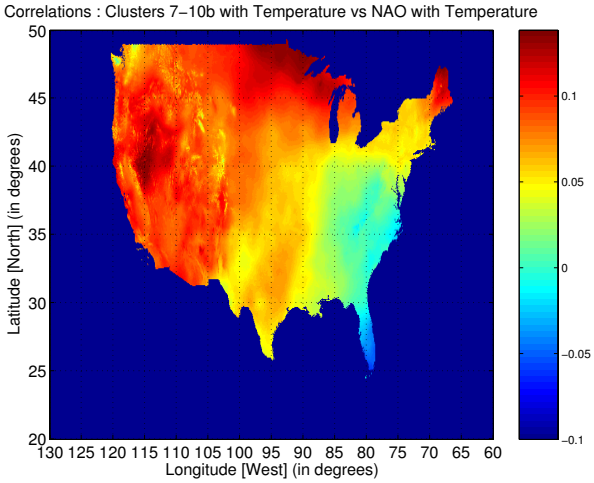


Figure 18: Comparison of best shifted correlation : Clusters 7-10b vs NAO

Similarly, we look at the differences for runs longer than 20, 40 and 80 days, and maximum length runs in Figures 19, 20, 21 and 22, respectively. The runs also show that our index performs as well as the standard index in most regions.

5. IMPACT OF SST ON LAND TEMPERATURE

In the second set of experiments we use the SST clusters discovered in [14]. Out of the 108 clusters, we selected the 28 that had area-weighted correlation comparable to known climate indices. In this section, we explore the possibility of using these clusters for predicting anomalies in global land temperature.

The SST and average temperature datasets consist of monthly

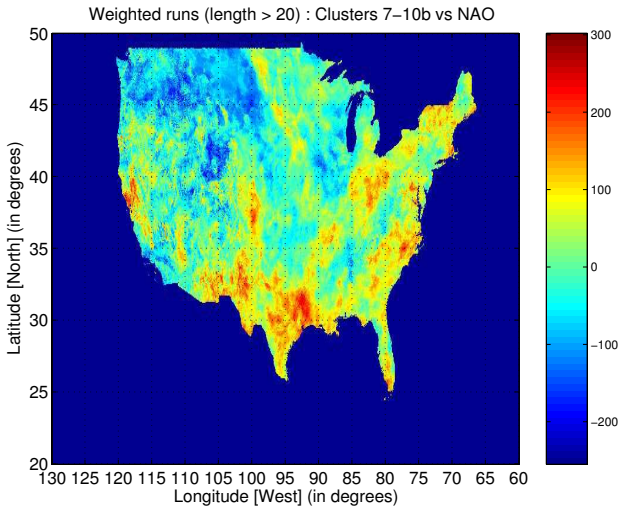


Figure 19: Comparison of Weighted runs of length 20 and more : Clusters 7-10b vs NAO

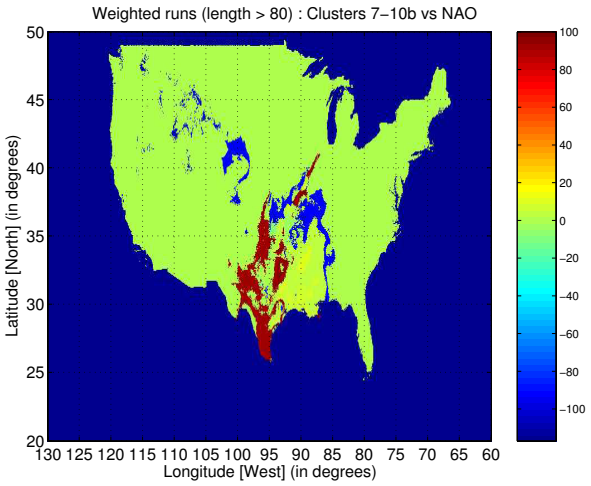


Figure 21: Comparison of Weighted runs of length 80 and more : Clusters 7-10b vs NAO

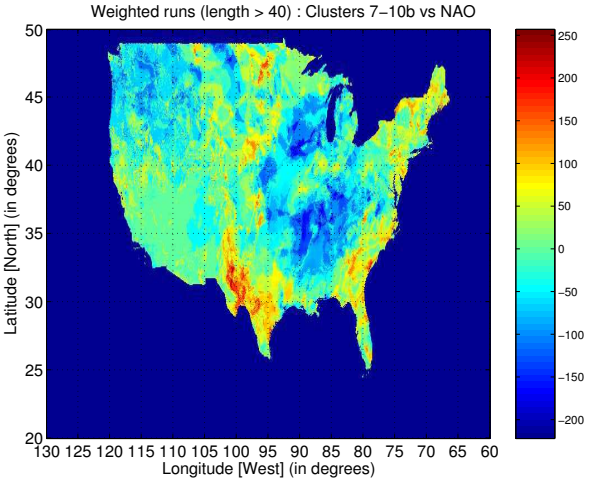


Figure 20: Comparison of Weighted runs of length 40 and more : Clusters 7-10b vs NAO

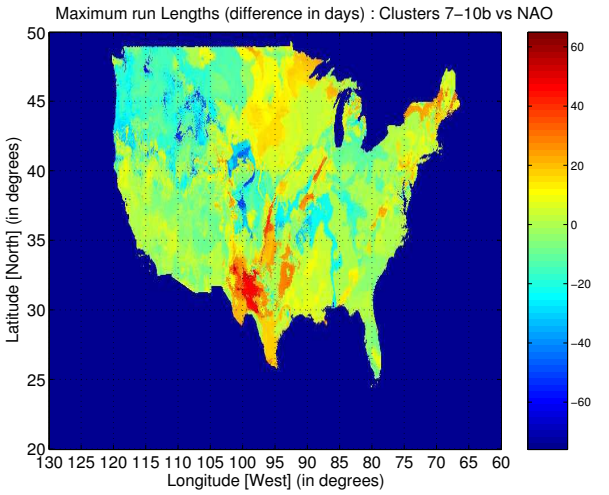


Figure 22: Comparison of longest runs (difference in days) : Clusters 7-10b vs NAO

measurements for the 41 years between 1958 and 1998. In order to remove seasonality, the temperature dataset was normalized on a monthly basis. This means that data for different months are normalized separately i.e., the 41 Januaries are normalized separately from the 41 Februaries. A positive temperature anomaly occurs, when the normalized temperature exceeds 1.2 standard deviations from the mean for that month (i.e., the mean of the 41 Januaries). Similarly, a negative anomaly occurs, when the normalized temperature falls below -1.2 standard deviations. Based on the SST time series (possibly in conjunction with the temperature time series), our goal is to predict whether a certain land region will have normal (not anomalous), anomalously high or low temperature in a given future month.

5.1 Methodology

First, we identify regions of the land that behave homogeneously. As explained earlier, significant climate processes affect larger areas, hence predictions that hold for larger areas are more reliable. We use SNN clustering to discover these regions because it has all the desired properties: the clusters are homogeneous and continuous. Figure 23 displays the 118 land clusters.

Next, for each land cluster and for each SST cluster, we build a model. The model is built on the first 20 years of the time series (between 1958 and 1977) using various predictors and the predictive performance of the model is evaluated on the remaining 21 years. Since both the positive and the negative anomalies are rare, we use F-measure as an indicator of the predictive performance of our model.

This set of experiments first tries to answer two questions: In which land areas can temperature be predicted by the selected 28 SST clusters? How do predictions based on these SST clusters compare to those obtained from (a)

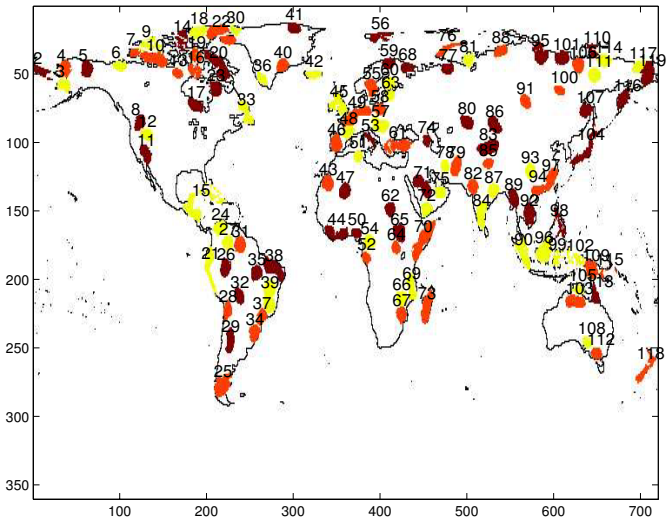


Figure 23: SNN clustering of the World based on land temperature

models that use known climate indices and (b) models that use only temporal autocorrelation? Second, the experiments also investigate whether the use of SST clusters can augment prediction based on temporal autocorrelation, and if so, how the results compare to predictions obtained using temporal autocorrelation augmented with known climate indices. (Temporal autocorrelation is useful for predicting time series, since, for example, today's temperature is often similar to yesterday's temperature. This approach also provides a baseline for comparison.)

5.2 Effect of SST Clusters on Land Clusters

For each land cluster and SST cluster pair, a predictive model is built. The predictors are the SST value of target month and the SST values for the preceding two months. Figures 25 and 28 show the land clusters plotted on a map and their colors are indicative of the F-measure with respect to the positive and negative anomalies.

For comparison purposes, Figures 24 and 27 depict the performance of temporal autocorrelation-based prediction. In this case, the predictors are the normalized temperatures for three consecutive months prior to the target month; naturally, not including the target month. Also, a comparison is made with the performance of predictions using the climate indices. The respective plots are provided in Figures 26 and 29.

Discussion In contrary to our expectation, the prediction results of autocorrelation based prediction were poor except for 3 clusters in South-America for positive anomalies and two cluster (one in East Africa and one in Australia) for negative anomalies.

The use of SST clusters or climate indices (instead of the temporal autocorrelation) appears to provide improved prediction results for most areas of the world. The known climate indices offer excellent predictive power of Peru and India for positive events and they predicted negative events well in South-America and Japan. SST clusters show improvement in predictive power in most of South-America and Indonesia for negative events, and in Scandinavia, South-Africa, South-America, Cambodia and Indonesia for positive events.

Some of the land regions, in which temperature can be predicted by climate indices and SST clusters, overlap. This was expected, because some of the climate indices are based on SST, and hence some of SST clusters were found to have high correlation with these climate indices.

It must be noted that the excellent performance of the known climate indices in Peru is due to the El-Niño indices. Since this work focuses on teleconnections, this particular prediction in Peru is not interesting, because the predictor lies very close to the shore. In order to avoid such misleading results, a distance based filtering was applied in case of the SST clusters: all prediction results where the distance between the predictor and the centroid of the land cluster is less than 3000 km were omitted. However, for climate indices, their locations are not always well defined [e.g., NAO is defined as the sea level pressure difference between two points], therefore such filtering could not be applied.

5.3 SST clusters enhancing auto-correlation based prediction

In this subsection, we investigate whether SST clusters can enhance the temporal auto-correlation based prediction. In this case, we use both the temperature data and the SST data as predictors: the temperature values for 3 months prior to the target month, and the SST values for three consecutive months including the target month. Figures 30 and 32 depict the prediction performance with respect to positive and negative anomalies. For comparison purposes, we have also substituted SST clusters with climate indices. Their performance is depicted in Figures 31 and 33.

Discussion. Prediction using auto-correlation in conjunction with climate indices or SST clusters produces considerably better prediction than either approach used alone. In the case of SST clusters, for negative anomalies, the improvement is substantial for portions of North- and South-America, the Middle East and Indonesia; for positive anomalies, for portions of South-America, Africa, India, Cambodia and Indonesia.

We detected a few regions, where the augmentation with climate indices had an adverse effect on predictive performance. We suspect that this phenomenon is simply a problem with our classifier, namely that the model overfits the data.

Generally, the improvement due to SST clusters is larger than the improvement due to the climate indices. For negative anomalies, climate indices have achieved larger improvement only in the Northern Arctic area, while improvements due to SST clusters are more pronounced in parts of South-America, Africa, the Mediterranean, the Middle East and Indonesia. For positive anomalies, the only two regions, where the climate indices have achieved higher improvement is Peru (due to the El-Niño indices) and Northern-Europe, while the improvement due to SST clusters was more marked in parts of North- and South-America, Africa, the Middle East and Indonesia.

6. CONCLUSIONS AND FUTURE WORK

In previous work we used clustering to discover potential climate indices. In this paper, we extended this work

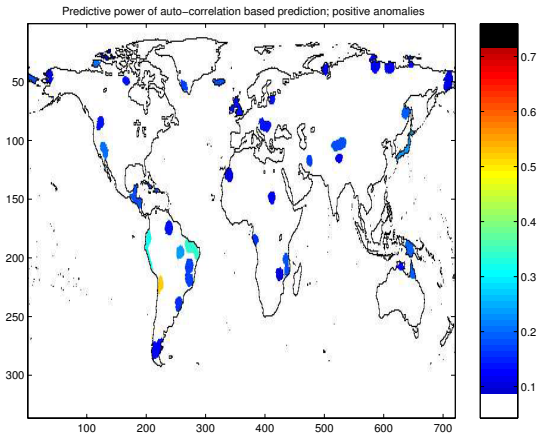


Figure 24: Prediction performance (F-measure) for *positive* anomalies using temporal auto-correlation.

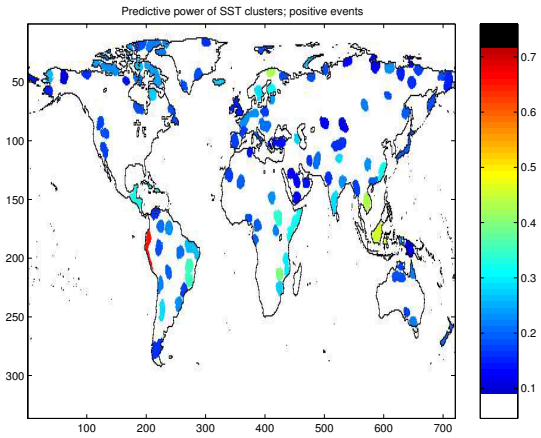


Figure 25: Prediction performance (F-measure) for *positive* anomalies using SST clusters.

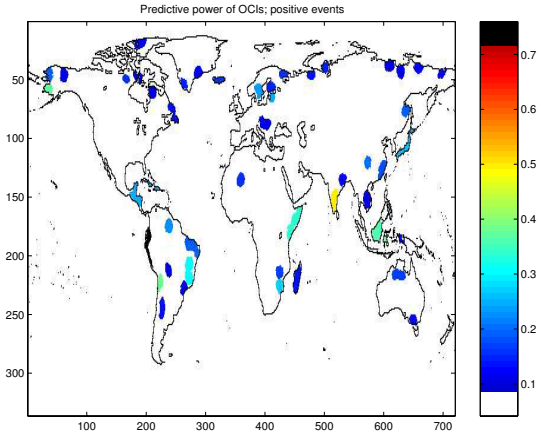


Figure 26: Prediction performance (F-measure) for *positive* anomalies using climate indices.

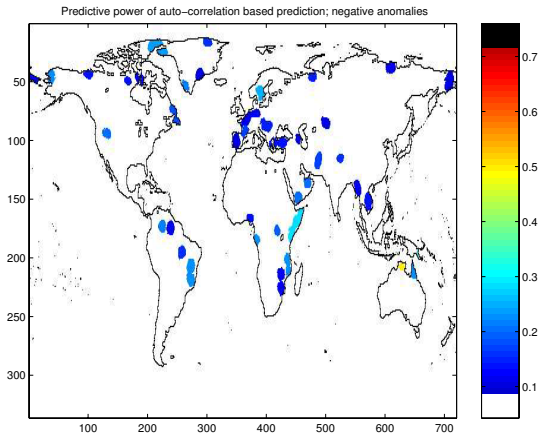


Figure 27: Prediction performance (F-measure) for *negative* anomalies using temporal auto-correlation.

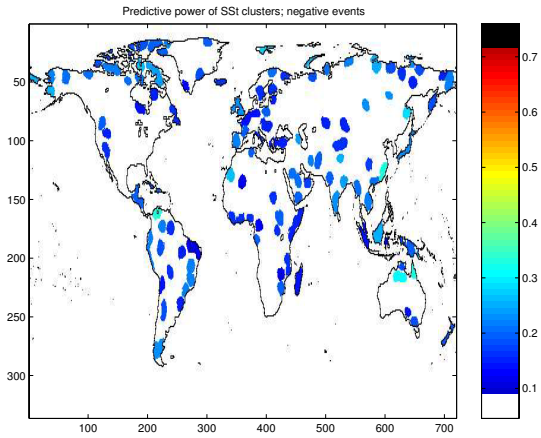


Figure 28: Prediction performance (F-measure) for *negative* anomalies using SST clusters.

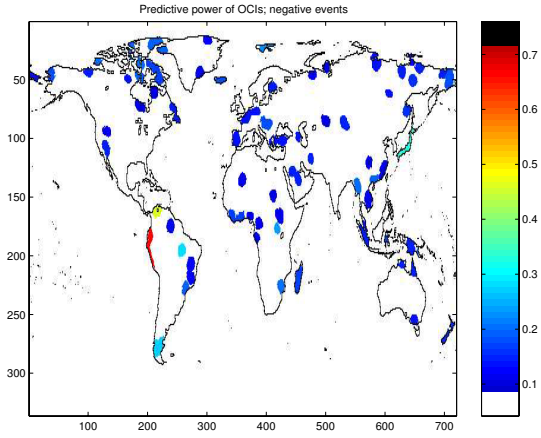


Figure 29: Prediction performance (F-measure) for *negative* anomalies using climate indices.

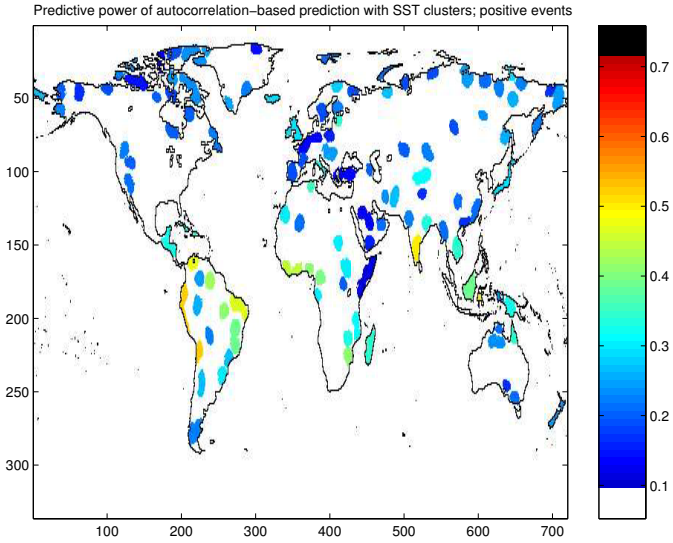


Figure 30: Prediction performance (F-measure) of predicting *positive* anomalies using auto-correlation in conjunction with SST clusters.

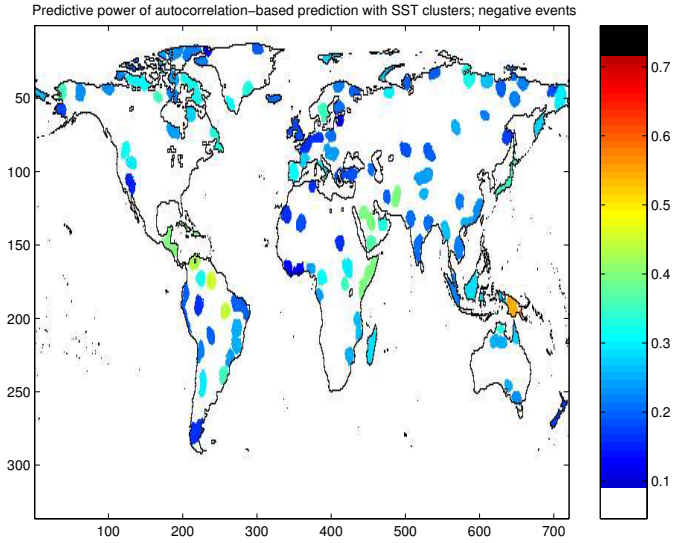


Figure 32: Prediction performance (F-measure) of predicting *negative* anomalies using auto-correlation in conjunction with SST clusters.

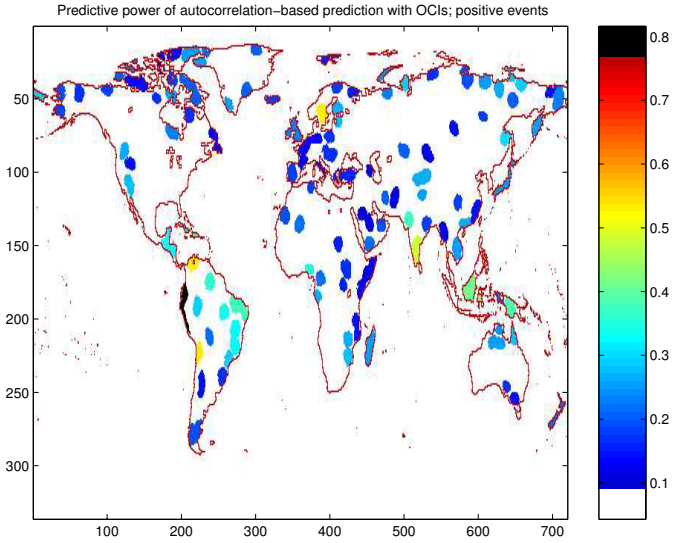


Figure 31: Prediction performance (F-measure) of predicting *positive* anomalies using auto-correlation in conjunction with known climate indices.

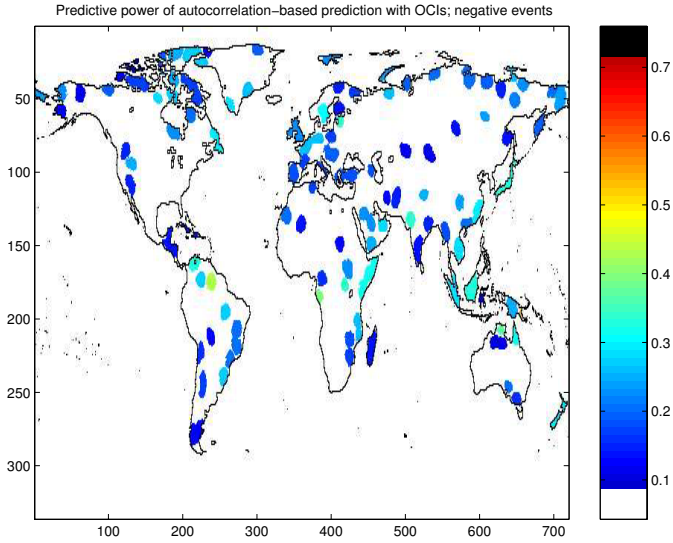


Figure 33: Prediction performance (F-measure) of predicting *negative* anomalies using auto-correlation in conjunction with known climate indices.

by exploring the feasibility of predicting land temperature using these cluster based climate indices. For one portion of this work, we generated a cluster-based index that was a variant of the well known NAO climate index and conducted experiments to compare the predictive performance of this index and NAO with respect to land temperature anomalies in the United States. We found that the cluster-based index performs as well as NAO in the Eastern US, but performs better in large portions of the western US. In another portion of this work, we used SST clusters as predictors of global land temperature anomalies and found, for certain regions of the world, that the SST clusters outperform known climate indices. We also showed that using SST clusters with autocorrelation-based prediction substantially improves prediction performance.

The results presented in this paper, while promising, are preliminary, and more work is needed in a number of areas. We need, for instance, to be better able to address the significance of the relationships that we find. For example, how likely is it that a high correlation between a land area and a cluster index can arise by chance? Similarly, how likely is it that the better predictive performance of cluster based

climate indices with respect to known climate indices is due to the fact that there are more cluster based climate indices than known climate indices? Also, we would like to explore predictive techniques other than Ripper and measures of similarity other than correlation. Finally, another important area for future work is 'moving' clusters, i.e., clusters whose locations change with time. In the experiments presented in this paper, we used stationary clusters, but moving clusters are expected to offer better results since they better model the underlying behavior of the Earth.

7. ACKNOWLEDGMENTS

This work was partially supported by NASA grant # NCC 2 1231, by NSF grant IIS-0308264, and by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAD19-01-2-0014. The content of this work does not necessarily reflect the position or policy of the government and no official endorsement should be inferred. Access to computing facilities was provided by the AHPCRC and Minnesota Supercomputing Institute.

8. REFERENCES

[1] J. V. Bradley. *Distribution-free Statistical Tests*. Prentice Hall, Englewood Cliffs, New Jersey, 1968.

[2] http://met-www.cit.cornell.edu/glossary.html.

[3] http://www.daymet.org.

[4] L. Ertöz, M. Steinbach, and V. Kumar. Finding topics in collections of documents: A shared nearest neighbor approach. In *Proceedings of Text Mine'01, First SIAM International Conference on Data Mining, Chicago, IL, USA*, 2001.

[5] L. Ertöz, M. Steinbach, and V. Kumar. A new shared nearest neighbor clustering algorithm and its applications. In *Workshop on Clustering High Dimensional Data and its Applications, SIAM Data Mining 2002, Arlington, VA, USA*, 2002.

[6] L. Ertöz, M. Steinbach, and V. Kumar. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of Third SIAM International Conference on Data Mining, San Francisco, CA, USA*, May 2003.

[7] J. W. Hurrell. Decadal trends in the north atlantic oscillation regional temperatures and precipitation. In *Science*, volume 269, pages 676–679, 1995.

[8] J. W. Hurrell, Y. Kushnir, G. Ottersen, , and e. M. Visbeck. *The North Atlantic Oscillation: Climatic Significance and Environmental Impact*. American Geophysical Union, 2003.

[9] http://www.cgd.ucar.edu/cas/catalog/climind/.

[10] http://www.cdc.noaa.gov/USclimate/Correlation/help.html.

[11] http://www.cpc.ncep.noaa.gov/products/precip/CWlink/daily_ao_index/history/history.html.

[12] http://www.cdc.noaa.gov/.

[13] D. H. Portis, J. Walsh, M. E. Hamly, and P. Lamb. Seasonality of the north atlantic oscillation. In *Journal of Climate*, volume 14, pages 2069–2078, 2001.

[14] M. Steinbach, P.-N. Tan, V. Kumar, S. Klooster, and C. Potter. Discovery of climate indices using clustering. In *KDD 2003*, 2003.

[15] M. Steinbach, P.-N. Tan, V. Kumar, C. Potter, S. Klooster, and A. Torregrosa. Clustering earth science data: Goals, issues and results. In *Proceedings of the Fourth KDD Workshop on Mining Scientific Datasets, San Francisco, California, USA*, August 2001.

[16] P.-N. Tan, M. Steinbach, V. Kumar, S. Klooster, C. Potter, and A. Torregrosa. Finding spatio-termporal patterns in earth science data. In *KDD Temporal Data Mining Workshop, San Francisco, California, USA*, August 2001.

[17] G. H. Taylor. Impacts of the el nino/southern oscillation on the pacific northwest. Technical report, Oregon State University, Corvallis, Oregon, 1998.

Density-based Clustering of Time Series Subsequences

Anne Denton
Department of Computer Science
North Dakota State University
Fargo, North Dakota 58105, USA
anne.denton@ndsu.nodak.edu

ABSTRACT

Doubts have been raised that time series subsequences can be clustered in a meaningful way. This paper introduces a kernel-density-based algorithm that detects meaningful patterns in the presence of a vast number of random-walk-like subsequences. The value of density-based algorithms for noise elimination in general has long been demonstrated. The challenge of applying such techniques to time-series data consists in first specifying uninteresting sequences that are to be considered as noise, and secondly ensuring that those uninteresting sequences will not affect the clustering result. Both problems are addressed in this paper and the success of the technique is demonstrated on several standard data sets.

1. INTRODUCTION

Frequent pattern mining algorithms are among the most important contributions of the data mining community to the data analysis toolbox. Frequent subsequences in time series data are of interest by themselves [13] and in combinations that form strong association rules [5]. Literature on association rule mining has largely assumed that time series subsequences can be encoded by letters through partitioning of the occurring subsequences using k-means clustering. This approach is motivated by its similarity to vector quantization [7] and can be justified through its property of minimizing the squared error function. It has been noted that the resulting cluster centers are, however, not very specific to the data set from which they originate [12]. This does not disqualify them from use in association rule mining since the technique does not require a data set dependent representation. It is, nevertheless important to determine if clustering can be used to mine for frequently occurring subsequences.

This paper demonstrates that kernel-density-based clustering [3, 9] is indeed capable of identifying cluster centers that are specific to the data set from which they originate. These cluster centers represent frequent subsequences and can, thereby, have applications similar to motifs [13]. The derivation based on the concept of a kernel-density has an important fundamental benefit: It will be shown in section 3 that normalization can make some sequences much more frequent than others, even for data that is created in a random fashion. It is, therefore, important to compare any result against the probability of getting that same result based on random input. The construction of a density landscape makes this comparison very easy.

Why, then, does it make a difference if density-based clustering is used instead of k-means or hierarchical clustering? Time series data often contain significant noise, i.e. subsequences that are the result of random fluctuations. Density-based clustering only considers those regions of the density-landscape as clusters that rise above a noise threshold. Setting the threshold appropriately can eliminate the impact of noise, provided the noise distribution leads to an approximately constant density surface. The noise tolerance of density-based clustering is extensively discussed in [9]. Adaptation to time series data requires a detailed understanding of what constitutes noise in that setting. Different time series models will be covered in section 2 to allow identification and targeted elimination of noise-like sequences.

The paper is organized as follows: Section 2 provides background on time series models and kernel-density-based clustering, section 3 describes the algorithm, section 4 demonstrates the results on real data, and 5 concludes the paper.

2. BACKGROUND

Many types of data are recorded as a function of time. This paper focuses on the analysis of a single sequence of real-valued data.

Definition 1: A time series $T = t_1, ..., t_n$ is a sequence of real numbers. Numbers correspond to values of an observed quantity, collected at equally spaced points in time.

Definition 2: A subsequence of time series $T = t_1, ..., t_n$, with length w, is a sequence $S = t_m, ..., t_{m+w-1}$ with $1 \leq m \leq n - w + 1$. The process of extracting subsequences by incrementing m in steps of one is called application of a sliding window. Subsequences will be represented as vectors in a w-dimensional vector space.

Clustering of any kind of data requires the definition of a similarity or of a distance measure. One of the best-known distance measures, and a popular choice in time series clustering, is the Euclidean distance. The Euclidean distance measure is a special case of an L_p norm. L_p norms may fail

to capture similarity well when being applied to raw time series data because differences in the average value and average derivative affect the total distance. Normalization is an important step to reduce this problem. In this paper Z-normalization is used in which the mean of the subsequence is subtracted and the data are divided by their standard deviation.

Other specialized distance measures have been described for time series clustering, such as dynamic time warping, DTW [2], and longest common subsequence similarity, LCSS [16]. They are not suitable to the short subsequences of interest in this paper.

2.1 Kernel-density-based Clustering

Clustering based on kernel-density estimation has been successfully used in many contexts [3, 4, 9]. Assuming n data points $x_i, i = 1, ..., n$ in a d-dimensional space, the kernel density estimator is given by

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \tag{1}$$

where the kernel function $K(x)$ is normalized

$$\int_{\mathbb{R}^d} K(\mathbf{x}) d\mathbf{x} = 1. \tag{2}$$

In this paper, a Gaussian kernel is used throughout

$$K^{(G)} = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{|\mathbf{x}|^2}{2}\right) \tag{3}$$

Since the Gaussian kernel is radially symmetric, a profile can be defined through

$$K(\mathbf{x}) = c_{k,d} k(|\mathbf{x}|^2) \tag{4}$$

where $c_{k,d}$ is a normalization constant that guarantees the normalization in equation (2). The goal of finding representative sequences is achieved by identifying the points in the density landscape that correspond to the highest density. In [9] these local maxima are referred to as density attractors and in [4] as modes. The modes are determined by picking starting points and updating their location through a hill climbing step. Following [4], updates, i.e. differences between tentative cluster center locations for successive steps, are computed as

$$\mathbf{m}_h(\mathbf{x}) = \frac{\sum_{n=1}^{n} \mathbf{x_i} g\left(\left|\frac{\mathbf{x}-\mathbf{x_i}}{h}\right|^2\right)}{\sum_{n=1}^{n} g\left(\left|\frac{\mathbf{x}-\mathbf{x_i}}{h}\right|^2\right)} - \mathbf{x} \tag{5}$$

where $g(x) = -k'(x)$ is the negative derivative of the kernel profile. Any data point can be associated with the cluster center to which it is attracted. Only modes above a threshold t are considered cluster centers. Data points that are attracted to modes below t are outliers or noise.

Clusters cannot only be defined based on the density attractors or modes but also as regions that are continuously above a threshold [9]. Such a definition allows multiple attractor regions to be joined into one arbitrarily-shaped cluster. In the case of time series subsequence clustering the focus lies on the identification of representative sequences and joining of attractor regions is thereby not considered helpful.

Kernel-density-based clustering is robust against noise, provided the noise leads to an approximately constant density surface. Constant contributions to the density distribution do not affect the position of maxima. The description of noise as uniformly distributed data points, which underlies the proof of noise invariance in [9], is not valid for typical time series subsequences. In the following it will be shown what distribution can be expected for time series data.

2.2 Time Series Models

Several time series models have been proposed [14]. We will now look for a model of a time series that describes noise in typical data sets well and can, thereby, serve as a noise definition.

The arguably simplest model of a time series is "strict white noise" as defined in [14], denoted by $\{e_t\}$. Mean, μ, and variance σ are assumed to be the same for all time points.

$$\begin{aligned} E\{e_t\} &= \mu, \quad \text{var}\{e_t\} = \sigma^2, \forall t, \\ &\quad \text{cov}\{e_t, e_s\} = 0, \forall t \neq s. \end{aligned} \tag{6}$$

Data that follow this distribution lead to a density landscape that is similar to that of random data in other subject domains. There will be a broad maximum at the value of μ in each dimension, which will be moved to the origin by the normalization. Kernel-density-based clustering identifies local maxima as clusters, provided they are higher than the broad maximum due to noise.

Time series data are not, however, typically described well by the model of "strict white noise". In the vast majority of settings we can expect some correlation between the values at successive points in time. For example, in stock market data the default assumption is for share values to remain constant. Nobody would consider resetting all expected share values to 0 or a constant value on a day without trading. Instead share values are assumed to approximately retain the value they had on the previous day of trading. There is correspondingly little benefit in eliminating "strict white noise". Clusters may still be determined by factors that are common to most time series and, thereby, reflect standard behavior rather than data-set-specific patterns. In this case clusterings could be similar or identical for different series and would still be considered meaningless according to [12].

A more advanced concept is the linear time series model in which future values are assumed to depend on past values as a linear superposition [14]

$$\sum_{u=0}^{\infty} h_u X_{t-u} = e_t \tag{7}$$

where the X_t are time series values, $\{h_u\}$ is a sequence of real values, and $\{e_t\}$ is a sequence of independent zero mean random variables, as defined in equation (6) with $\mu = 0$. In the simplest case, the next value of the time series depends only on one current value. A process in which future states only depend on the present state is called a Markov process. Assuming further that the expression on the left hand side of equation (7) is simply the difference between successive steps, we arrive at a time series model that qualifies as a good example of a random sequence in most settings. In

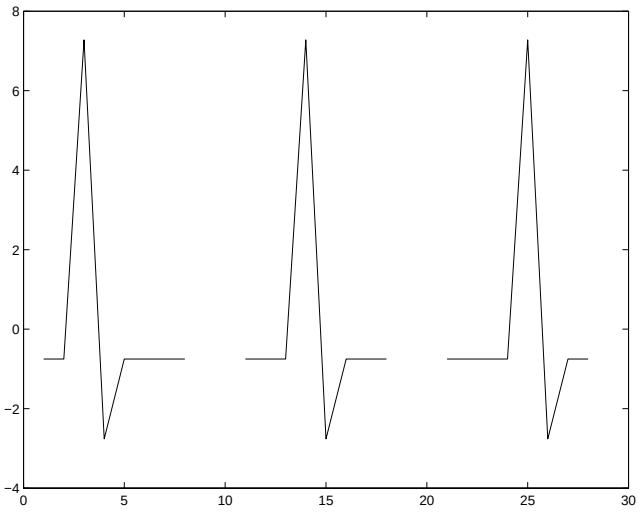


Figure 1: Subsequences in which a characteristic pattern occurs in different locations.

stock data, for example, random fluctuations would neither give us reason to buy nor to sell stock. Deviations from such behavior, on the other hand, are considered interesting patterns. Such a time series is also called a random walk time series.

2.3 Random Walk Time Series

The concept of a random walk was introduced to describe particle motion in n dimensions [15]. For time series data time advances regularly and the random walk is restricted to one dimension, corresponding to the values the time series can take.

Definition 3: A Gaussian random walk time series is a normally distributed sequence that satisfies

$$\begin{aligned} &X_t - X_{t-1} = e_t \\ &E\{e_t\} = 0, \quad \text{var}\{e_t\} = 1, \ \forall t, \\ &\text{cov}\{e_t, e_s\} = 0, \ \forall t \neq s. \end{aligned} \tag{8}$$

The random walk time series that is used as an example series to evaluate the effectiveness of the algorithm was taken from [11] and follows this model. Note that the choice of mean and variance are irrelevant in this definition because subsequences are normalized.

A slightly modified version of this concept is used to derive a random walk space that can be completely enumerated. It is then necessary to limit the number of possible random values to a finite number, in this case 1 and -1. Such a simplification is also common for random walks in their traditional use [15].

Definition 4: A discrete random walk time series is a sequence that satisfies

$$\begin{aligned} &X_t - X_{t-1} = r_t \\ &E\{r_t\} = 0, \quad |r_t| = 1, \forall t, \\ &\text{cov}\{r_t, r_s\} = 0, \ \forall t \neq s. \end{aligned} \tag{9}$$

Definition 5: The space of all discrete random walk time series of a given length is called random walk space.

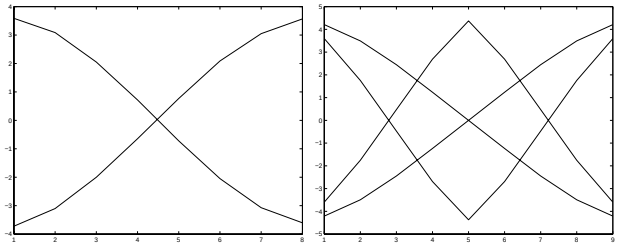


Figure 2: Density-based clustering of a Gaussian random walk (left) and a persistent discrete random walk (right).

Finally, it can be observed that for many time series not only the values of successive points are correlated, but also the slopes between them, i.e., if there is an increase between time $t-1$ and time t there is more likely to be an increase than a decrease from t to $t+1$. Random walks with such correlations are called persistent. While not all time series are persistent, those that are, should be treated with an appropriate noise model. Patterns that are entirely due to persistence are not expected to be of interest to a user because they are independent of the specific data set. The same patterns would be observed for all data sets with the same level of persistence.

Definition 6: A persistent random walk time series is a sequence that satisfies

$$\begin{aligned} &X_t - X_{t-1} = r_t \\ &E\{r_t\} = 0, \quad |r_t| = 1, \ \forall t, \\ &\text{cov}\{r_{t-1}, r_t\} = c, \ \forall t. \end{aligned} \tag{10}$$

where c is a constant that is determined from the data. If the time series of interest does not show persistence, this model reduces to equation (9).

3. APPLICATION OF KERNEL-DENSITY-BASED CLUSTERING TO TIME SERIES

Kernel-density-based clustering can be implemented in two fundamentally different ways. Data points can be stored in a table that is parsed once for each iteration of the hill-climbing algorithm [3, 4]. Alternatively a grid representation can be used within the space that is spanned by all attributes [9]. In the latter approach the volume of the space increases exponentially with the number of dimensions of the data. The former approach was therefore taken to achieve acceptable scaling to large subsequences. The implementation was done in MATLAB as an extension of [10], a kd-tree-based [1] MATLAB toolbox for kernel-density estimation. Modes are evaluated using a hill-climbing algorithm. Starting points are all data points that are larger than their d nearest neighbors within the data set, where d is the number of dimensions (points in each subsequence). This strategy was tested against the more rigorous choice of picking each data point as starting position. Results showed that all modes were reliably identified and the speed was significantly increased by limiting the set of starting points. Comparison of each point with its neighbors is very fast given the kd-tree representation.

Further modifications were necessary to adapt the density-

```
 1  length = 8 % length of subsequences
 2  sigma = 2.1% variance of Gaussian in Kernel Density Estimation
 3  threshold = 0.3e-4 % density threshold
    % Prepare subsequences
 4  dataArray = normalize(slidingWindow(sequence))
    % Prepare random walk representation
 5  rwLibrary= createRandomWalks(length)
 6  rwArray = closestRandomWalk(dataArray,rwLibrary)
 7  ratio = successiveSlopesSameVsDifferent(rwArray)
 8  rwKernelDensity = densityOfRws(length,sigma,ratio)
    % Calculate weights to compensate for random walk density
 9  rwKernelDensityMatrix = evaluate(rwKernelDensity,rwLibrary)
10  weights = conjugateGradient(rwKernelDensityMatrix,vectorOfOnes)
11  weights = applyLowerBound(weights,lowerBound)
12  weightArray = mapToData(rwArray,weights)
    % Actual clustering step using kernel density estimation
13  dataKernelDensity = kde(dataArray,sigma,weightArray)
14  start = getMaxima(dataKernelDensity,length)
15  [clusterCenters,centerDensities] = modes(dataKernelDensity,start)
16  clusterCenters = filterCloseCenters(clusterCenters)
17  clusterCenters = applyThreshold(clusterCenters,centerDensities,
    threshold)
```

Figure 3: Pseudocode

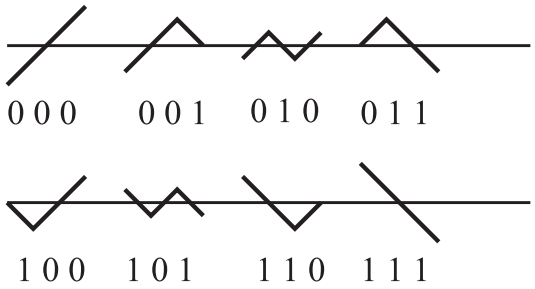


Figure 4: Random walk space for a time series of length 4.

based clustering algorithm to time series subsequences. One problem is that a characteristic pattern that is shorter than the window size will result in several cluster centers. Figure 1. illustrates the problem. A clear pattern of a large data point followed by a small one can be identified in all three sequences. The sliding window approach samples data points multiple times and can thereby lead to multiple representations of the same pattern from the original time series. A user is likely to only be interested in this pattern once. The problem was resolved as a post-processing step in which potential cluster centers are compared and ones that are closer to each other than a threshold are eliminated. Sequences are shifted and only those parts compared that are defined for both subsequences. The resulting sequences are normalized and the distance weighted (using a weight of $\sqrt{(}i+1)$ for a shift of i) to give penalty to shifted matches. Shifts up to half the length of the time series subsequence are considered.

Kernel-density-based clustering, as described so far, leads to non-trivial maxima in the density landscape even if the input sequences are perfect random walks. Figure 2. shows the result of density-based clustering of a Gaussian random walk (left) and a persistent discrete random walk (right). The next section will show how to compensate for this uninformative result.

3.1 Density Distribution of a Random Walk and its Inversion

The idea of the algorithm is as follows: The density landscape of discrete random walk data is evaluated. Input data is then weighted such that random walk data would lead to a constant density surface. Handling this problem computationally requires that only a finite number of sequences are used to define the random walk space. The discrete random walk definition in equation (9) is used for this purpose. All random walks that are possible according to this definition can be constructed in a straight forward way. Figure 4. shows the space of random walks for sequences of length 4. For any two successive data points there is a choice of an increase or decrease by 1 (3 choices for 4 data points). In general, the number of discrete random walks that have to be considered for sequences of length w is $2^{(w-1)}$. Each possible walk can be efficiently represented by a $(w-1)$-digit binary number that serves as an index for array representations of density and other properties.

Subsequences of real data, naturally, do not match these sequences exactly. Two possible prescriptions of identifying corresponding random walks for a given data sequence were evaluated. Each normalized data sequence can be compared with all possible normalized random walk sequences and the closest chosen to represent the sequence. Alternatively a difference-based matching process can be used that considers only the sign of the slope between neighboring points in the sequence. The latter approach shows better scaling to large sequences but does not capture the relevant features of the data sequences as well. The first approach was, therefore,

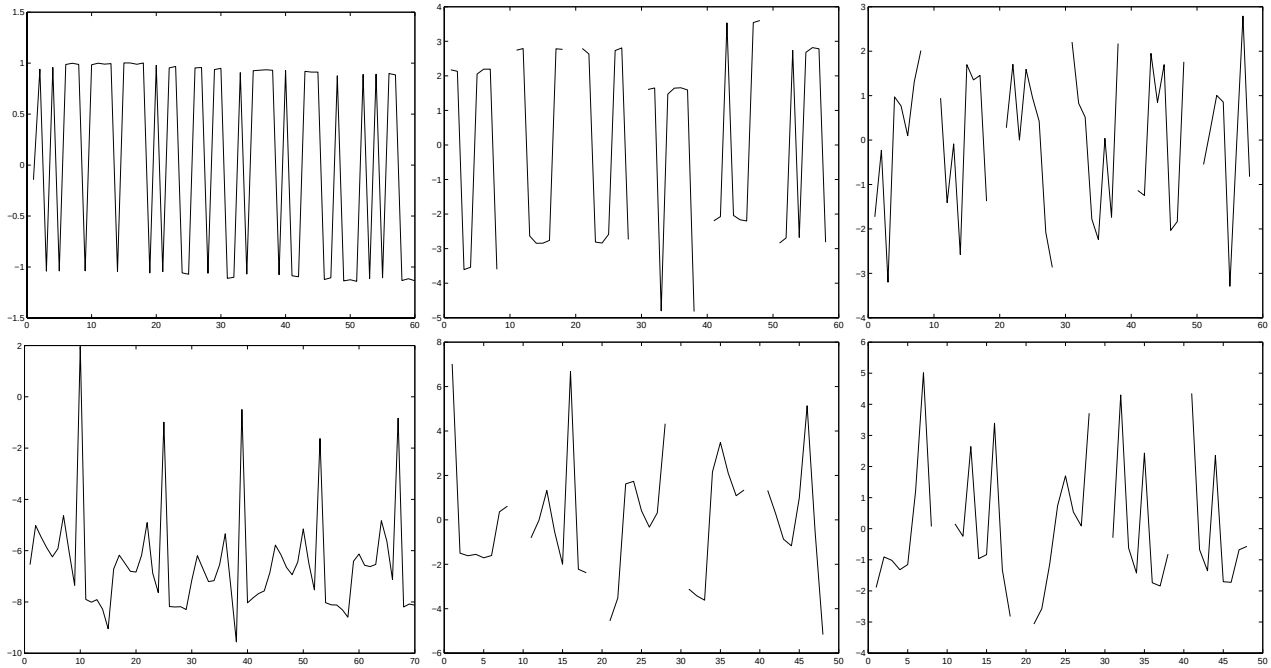


Figure 5: Data set glassfurnace (top) and ecg (bottom); shown are the time series itself (left), results of density-based clustering (middle), and results of k-means clustering (right).

taken.

Persistence, as defined in equation (10), is incorporated based on the statistics of the data. After representative random walk sequences have been picked, the number of occurrences of equal slopes is determined. The ratio of equal over different slopes is calculated. Based on that ratio, all random walks in the random walk space are weighted. A density landscape is constructed with the same kernel function as is used for the data.

The goal of calculating the density landscape of random walks is to compensate for any patterns that occur by random sequences alone. Weights are determined for each random walk model sequence in the random walk space. For each real sequence the weight of the corresponding model sequence is supplied as part of the construction of the density landscape. Weights are defined such that the weighted density landscape based on the set of model sequences is 1 for the location of each sequence. Determining the weights amounts to solving the following equation for $\mathbf{x}$

$$\mathbf{1} = \mathbf{K}^{(G)}\,\mathbf{x} \tag{11}$$

where $\mathbf{1}$ is a vector with all elements equal to one, and $\mathbf{K}^{(\mathbf{G})}$ is the kernel matrix for a Gaussian kernel

$$K_{i,j}^{(G)} = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{|\mathbf{x}_i - \mathbf{x}_j|^2}{2}\right) \tag{12}$$

evaluated at the positions of each model sequence, i.e., each point in the random walk space. In principle, the matrix $\mathbf{K}^{(\mathbf{G})}$ could be inverted. Since equation (11) only has to be solved once, and an approximate solution is satisfactory, a biconjugate gradient method as provided in MATLAB (bicgstab) is used instead.

The resulting vector $\mathbf{x}$ can – and commonly does – show negative values. This result is not surprising since equation (11) is not guaranteed to have a solution, for which all elements of $\mathbf{x}$ are positive. In practice it is, however, questionable if input sequences should be weighted with a negative weight. Doing so would mean that the existence of a particular sequence in the data set leads to a decrease in the density landscape. The minimum acceptable value for elements of $\mathbf{x}$ was set to be 0.01 times the maximum value occurring in $\mathbf{x}$. All smaller values were set to that minimum value.

3.2 Summary of the Algorithm

Figure 3. summarizes the steps in the algorithm. As a first step subsequences are extracted using a sliding window and are normalized, step 4. Several different normalization techniques were evaluated such as using the derivative of sequences as discussed in [6]. As a further alternative the original subsequences were taken, the mean subtracted and the standard deviation of differences between successive points was used for normalization, compare equation (8). Over all, Z-normalization led to the best performance and was chosen.

The next steps 5.-12. deal with the creation of a random walk representation, steps 5.-8., and the creation and approximate inversion of the kernel matrix, steps 9.-12. These steps are specific to the time series subsequence setting and are not used in the standard and adapted density-based clustering variants in section 4.

The clustering steps 13.-16. are implemented as outlined in section 2.1. Note that this algorithm may still return maxima as a result of noise in the form of random sequences. Due to the somewhat artificial nature of the reference sequences as discrete persistent random walks, Gaussian ran-

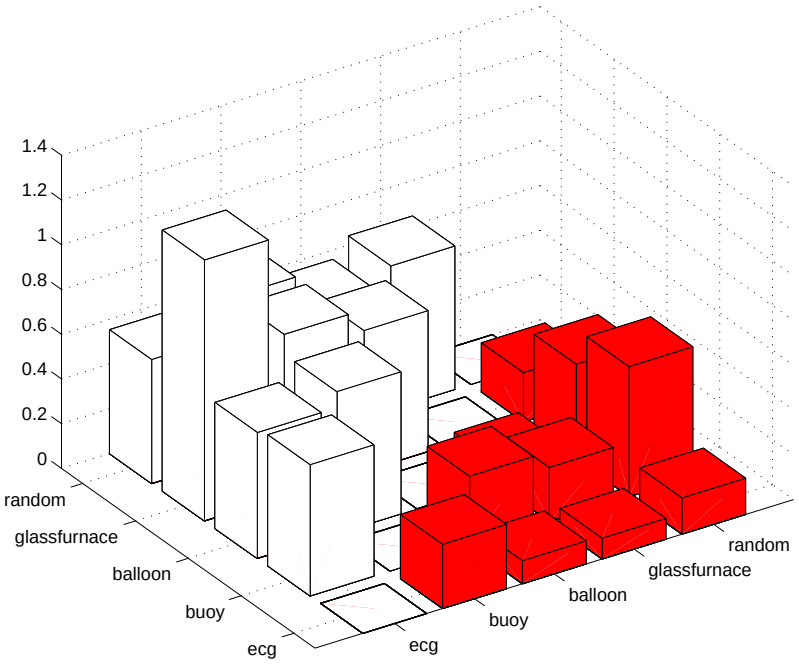


Figure 6: Comparison of density-based (dark bars, right half) with k-means clustering (white bars, left half).

dom walks will still result in systematic maxima similar to the left part of figure 2., albeit with a much lower density. The lower bound on weights also contributed an uneven density surface for random walks. Density values of clusters that were evaluated based on Gaussian random data were used as a guideline to determine a suitable threshold, below which maxima are not considered as cluster centers. Step 17. ensures that only clusters that have a density higher than that threshold are returned.

4. EXPERIMENTAL EVALUATION

The algorithm was evaluated on several standard data sets from the UCR Time Series Data Mining Archive [11] as well as on an ecg series (MIT-BIH Arrhythmia Database: mitdb100) from PhysioBank [8]. The ecg series was compressed by averaging over 20 consecutive values, the buoy series from the UCR Archive by averaging over 4 values. All experiments were done using subsequences of length 8. The width of the Gaussian kernel was chosen to be 2.1 and the density threshold for cluster centers 0.3×10^4.

Figure 5. (top) shows a data set from [11] together with the clustering results from density-based and k-means clustering. The cluster centers from density-based clustering clearly represent the typical patterns in the time series. Cluster centers from k-means clustering, on the other hand, bear no resemblance to the original data. One may argue that this time series has a rather extreme shape. Figure 5. (bottom) shows the same comparison for ecg data. Although the contrast is not as extreme, it can again be clearly seen that most of the k-means cluster centers do not represent any particular part of the sequence. No parts of that ecg sequence show two similarly high, pointed maxima next to each other. Yet, three of the k-means clusters that show such subsequences. All density-based clusters can be identified with subsequences of the original time series.

4.1 Meaningfulness of Clusterings

Following the idea in [12] the meaningulness of clustering is evaluated. For this purpose data sets were broken in half, and the cluster centers derived from both halves were compared using the meaningfulness-measure

$$\begin{aligned} &\text{cluster\_distance\_n(A, B)} \\ &\quad \equiv \frac{1}{k_A} \sum_{i=1}^{k_A} \min \left[dist(\overline{a}\_i, \overline{b}\_j) \right], \quad 1 \le j \le k_B \end{aligned} \tag{13}$$

$$\begin{aligned} &\text{clusteringmeaningfulness(X, Y)} \\ &\quad \equiv \frac{\text{within\_set\_X\_distance\_n}}{\text{between\_set\_X\_and\_Y\_distance\_n}} \end{aligned} \tag{14}$$

where $A = (\overline{a}_1, \overline{a}_2, ..., \overline{a}_{k_A})$ are the cluster centers derived from the first half of the data set and $B = (\overline{b}_1, \overline{b}_2, ..., \overline{b}_{k_B})$ are the cluster centers derived from the second half. This test is stricter than the original test that was based on separate runs of the k-means algorithm on the same data. Since density-based clustering has no random component the original test could not be applied. Since clusterings could differ in the number of clusters the cluster_distance measure had to be normalized with the number of sequences.

The average number of clusters in density-based clustering, averaged over all data sets listed in table 3.2., was approximately 4 (4.3). Comparisons are, therefore, done with k-means clustering, with $k = 4$. Only data sets were used that showed cluster centers with densities higher than clusters in random data. This criterion excluded some popular data

Table 1: Meaningfulness Results for Different Combinations of Data Sets

	mean (excl. random)		buoy	balloon	glassfurnace	random
Weighted Density-based Clustering	0.189	ecg	0.29	0.10	0.09	0.16
		buoy		0.31	0.24	0.58
		balloon			0.10	0.42
		glassfurnace				0.22
Adapted Density-based Clustering	0.228	ecg	0.26	0.15	0.11	0.08
		buoy		0.52	0.15	0.63
		balloon			0.18	0.29
		glassfurnace				0.06
Standard Density-based Clustering	0.390	ecg	0.18	0.21	0.46	0.10
		buoy		0.47	0.41	0.63
		balloon			0.60	0.36
		glassfurnace				0.34
K-means Clustering	0.718	ecg	0.59	0.56	1.17	0.55
		buoy		0.63	0.72	0.72
		balloon			0.63	0.56
		glassfurnace				0.64

sets used in other works such as the Standard and Poor 500 stock index. With the current sensitivity of the algorithm it is not possible to distinguish between the stock index and random walk data.

Figure 6. shows the results of density-based clustering with compensation for a persistent random walk density distribution (right, dark) together with results from k-means clustering (left, white). It can clearly be seen that the density-based results lead to much smaller values corresponding to more meaningful results.

It can also be seen that the difference between k-means and density-based clustering is not as significant for comparisons with random walk data. This is expected since the similarity between runs on different random walk sequences should not be similar. That means that the within_set_X_distance_n for random data should be as high as the between_set_X_and_Y _distance_n to other sequences. A meaningfulness value of about 0.5 should, therefore, be expected for any calculation involving random data. The fact that meaningfulness values are rather lower is an indication that cluster centers for random data still do have components of the systematic results in figure 2. because the model was built on discrete rather then continuous random walks, and weights were restricted to be above a threshold.

Table 3.2. shows detailed results for the runs in fig. 6. as well as for two other variants of density-based clustering: Standard density-based clustering uses the modes finding algorithm from [10] without modifications, and adapted density-based clustering allows shifting of subsequences in comparisons (see section 2.3).

5. CONCLUSIONS

Kernel-density-based clustering was successfully adapted to time series subsequence data. It was, thereby, shown that time series subsequence clustering can lead to meaningful results. A random walk space was introduced as a means of defining a reference distribution of sequences. Persistence of random walks was included in the treatment. The reference distribution was then used to derive weights for time series subsequence data. These weights compensated for the uneven distribution of random walk data. The resulting algorithm was evaluated on several standard data sets.

6. REFERENCES

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9), 1975.

[2] D. Berndt and J. Clifford. *Advances in knowledge discovery and data mining*, chapter Finding patterns in time series: a dynamic programming approach, pages 229–248. AAAI Press, Menlo Park, CA, 1996.

[3] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995.

[4] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[5] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *Proceedings of the IEEE Int. Conf. on Data Mining*, Rio de Janeiro, Brazil, 1998.

[6] M. Gavrilov, D. Anguelov, P. Indyk, and R. Motwani. Mining the stock market (extended abstract): which measure is best? In *Sixth ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 487–496, Boston, MA, 2000.

[7] A. Gersho and R. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Boston, MA, 1992.

[8] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000 (June 13).

[9] A. Hinneburg and D. Keim. A general approach to clustering in large databases with noise. *Knowl. Inf. Syst.*, 5(4):387–415, 2003.

[10] A. Ihler. Kernel density estimation toolbox for matlab (r13), accessed 04/2003.
[11] E. Keogh and T. Folias. The ucr time series data mining archive, 2002.
[12] E. Keogh, J. Lin, and W. Truppel. Clustering of time series subsequences is meaningless: implications for previous and future research. In *Proceedings of the IEEE Int. Conf. on Data Mining*, pages 115–122, Melbourne, FL, 2003.
[13] P. Patel, E. Keogh, J. Lin, and S. Lonardi. Mining motifs in massive time series databases. In *Proceedings of the IEEE Int. Conf. on Data Mining*, Maebashi City, Japan, 2002.
[14] M. Priestley. *Non-linear and non-stationary time series analysis*. Academic Press, 1988.
[15] F. Reif. *Fundamentals of Statistical and Thermal Physics*. McGraw-Hill, New York, NY, 1965.
[16] M. Vlachos, D. Gunopoulos, and G. Kollios. Discovering similar multidimensional trajectories. In *Proceedings 18th International Conference on Data Engineering (ICDE'02)*, San Jose, CA, 2002.

Clustered segmentations

Aristides Gionis Heikki Mannila Evimaria Terzi

HIIT, Basic Research Unit
Department of Computer Science
University of Helsinki, Finland
lastname@cs.helsinki.fi

ABSTRACT

The problem of sequence and time-series segmentation has been discussed widely and it has been applied successfully in a variety of areas, including computational genomics, data analysis for scientific applications, and telecommunications. In many of these areas the sequences involved are multidimensional, and the goal of the segmentation is to discover sequence segments with small variability. One of the characteristics of existing techniques is that they force all dimensions to share the same segment boundaries, yet, it is often reasonable to assume that different dimensions are more correlated than others, and that concrete and meaningful states are associated only with a subset of dimensions.

In this paper we study the problem of segmenting a multidimensional sequence when the dimensions of the sequence are allowed to form clusters and be segmented separately within each cluster. We demonstrate the relevance of this problem to many data-mining applications. We discuss the connection of our setting with existing work, we show the hardness of the suggested problem, and we propose a number of algorithms for its solution. Finally, we give empirical evidence showing that our algorithms work well in practice and produce useful results.

1. INTRODUCTION

Methods for segmenting sequence and time-series data have been discussed widely in the area of data mining. The goal of the segmentation problem is to discover sequence segments with small variability, and segmentation algorithms have been a key to compact representation and knowledge discovery in sequential data. A variety of segmentation algorithms have been proposed and they have been used successfully in many application areas, including computational genomics, data analysis for scientific applications, and telecommunications [8, 10, 14, 17]. In the next paragraphs we discuss three concrete examples in which different kinds of segmentation methods have been applied effectively; these examples also motivate the work presented in this paper.

Example 1. Himberg et al. [10] demonstrated the applicability of sequence-segmentation algorithms for the problem of "context awareness" in the area of mobile communications. The notion of context awareness can be a very powerful cue for improving the friendliness of mobile devices. As a few examples consider the situations where a mobile device adjusts automatically the ring tone, the audio volume, the screen font size, and other controls depending on where the users are located, what they are doing at that time, who else is around, what is the current noise or temperature level, and numerous other such context variables. The approach followed by Himberg et al. [10] is to infer context information from sensors attached to mobile devices: sensors for acceleration, noise level, temperature, luminosity, etc. Measurements from these sensors form naturally multidimensional time series. "Context" can then be inferred by segmenting the time series and annotating the various segments with concrete state descriptions (for example, if ACCELERATION $\approx u_0$, NOISE $\approx v_0$, and ILLUMINATION $\approx w_0$, then STATE = WALKINGINTHESTREET).

Example 2. The problem of discovering *recurrent sources* in sequences is examined in [8]. The idea is that many genomic (or other multivariate) sequences are often assembled by a small number of possible "sources", each of which might contribute several segments in the sequence. For instance, Azad et al. [2] try to identify a coarse-grained description of a given DNA string in terms of a smaller set of distinct domain labels. The work in [8] extends existing sequence-segmentation algorithms in a way that the resulting segments are associated with a description label, and the same label might appear in several segments of the sequence.

Example 3. One of the most important discoveries for the search of structure in genomic sequences is the "block structure" discovery of haplotypes. To explain this notion, consider a collection of DNA sequences over n marker sites (e.g., SNPs) for a population of p individuals. The "haplotype block structure" hypothesis states that the sequence of markers can be segmented in blocks, so that, in each block most of the haplotypes in the population fall into a small number of classes. The description of these haplotypes can be used for further knowledge discovery, e.g., for associating specific blocks with specific genetic-influenced diseases [9].

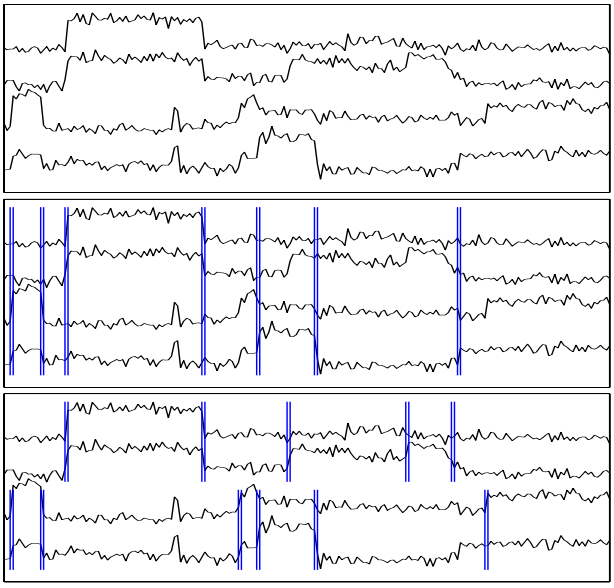


Figure 1: An illustrating example.

From the computational point of view, the problem of discovering haplotype blocks in genetic sequences can be viewed as partitioning a long multidimensional sequence into segments, such that, each segment demonstrates low diversity among the different dimensions. Naturally, segmentation algorithms have been applied to a good effect for this problem [17, 6, 20, 22].

In all of the above examples, segmentation algorithms have been employed for discovering the underlying structure of multi-dimensional sequences. The goal is to find segments with small variability. In previous approaches, however, dimensions are typically forced to share the same segment boundaries. On the other hand, it is often reasonable to assume that different dimensions are more correlated than others, and that concrete and meaningful states are associated with only small subsets of the dimensions.

An illustrating example of the above assumption is shown in Figure 1. The input sequence, a four-dimensional time series, is shown in the top box. In the middle box, it is shown a globally optimal segmentation for the input time series when all dimensions share common segment boundaries. In this example, one can see that the global segmentation provides a good description of the sequence—in all dimensions most of the segments can be described fairly well using a constant value. However, some of the segments are not quite uniform, while on the other hand there are some segment boundaries introduced in relatively constant pieces of the sequence. These representation problems can be alleviated if one allows different segment boundaries among subsets of dimensions, as shown in the lower box of Figure 1. We see that the segmentation after clustering the dimensions in pairs $\{1, 2\}$ and $\{3, 4\}$ gives a "tighter" fit and a more intuitive description of the sequence, even though the number of segments used for each cluster is smaller than the number of segments used in the global segmentation.

In this paper we study the problem of segmenting a multidimensional sequence when subsets of dimensions are allowed to be clustered and segmented separately from other subsets. We call this problem *clustered segmentation*.

Clustered segmentation can be used to extend and improve the quality of results in all of the applications mentioned in our motivating examples: In the application of context awareness, certain context states might be independent of some sensor readings, therefore, a segmentation based on all sensors simultaneously would be a bad predictor. For the problem of discovering recurrent segments one can imagine situations that sources are associated only with a subset of dimensions, for instance, specific regions in DNA sequences (CpG-islands) are associated with a low-dimensional signal (concentration of C and G bases) out of the whole genomic "vocabulary". Finally, in the problem of haplotype-block discovery, the sequence dimensions correspond to different individuals of the population, and thus, clustering of the dimensions would allow the discovery of subpopulation groups with distinct haplotype block structure. The existence of such subpopulations gives rise to a *mosaic structure* of haplotypes, which is a viable biological hypothesis [23, 25].

For solving the problem of clustered segmentation, measures and algorithms that give more refined segmentations and describe the data accurately need to be found. In this paper we are making the following contributions:

- We define the problem of clustered segmentation, and we demonstrate its relevance with existing data-mining applications.
- We demonstrate the hardness of the clustered segmentation problem and we develop practical algorithms for its solution.
- We perform extensive experimental evaluation on synthetically generated and real data sets. Our experiments study the behavior of the suggested algorithms. Furthermore, for many cases of real data sets, we show that clustered segmentation can provide a better model for representing and understanding the data.

The rest of this paper is organized as follows. In the next Section we define the problem of clustered segmentation in detail and we discuss its connection with other known problems. In Section 3 we describe a number of algorithms for solving the problem of clustered segmentation, and in Section 4 we describe our experiments. Finally, Section 5 is a short conclusion.

2. DESCRIPTION OF THE PROBLEM

As we discussed in the introduction, the goal of this paper is to develop improved measures and algorithms for the problem of multi-dimensional sequence segmentation. We now describe the problem in detail by first introducing the necessary notation. Let $S = \{s_1, \ldots, s_d\}$ be a d-dimensional sequence, where s_i is the i-th dimension (signal, attribute, individual, etc.). We assume that each dimension is a sequence of n *values*, and we denote by $s_i[u]$ the value at the u-th *position* of s_i. We write $S' \sqsubseteq S$ to denote that S' is a

subsequence of S, and $s_i[u, v]$ is the subsequence of s_i between the positions u and v. Similarly we denote by $S[u, v]$ the subsequence of S between the positions u and v when all dimensions are considered. The positions on each dimension are naturally *ordered*, for example, in time-series data the order is induced by the time attribute, while in genomic data the order comes from the position of each base in the DNA sequence. In addition, we assume that the dimensions of the sequence are *aligned*, that is, the values at the u-th position of all dimensions are semantically associated (e.g., they correspond to the same time).

We denote by σ a *k-segmentation* of the sequence $\langle 1, \ldots, n\rangle$, that is, a partitioning of $\langle 1, \ldots, n\rangle$ into k contiguous and non overlapping segments. As a result, for specifying a k-segmentation it suffices to provide the $k-1$ boundary points. We write a k-segmentation as $\sigma = \langle \sigma[1], \ldots, \sigma[k]\rangle$, so that we can refer to the individual segments. Because $\sigma[t]$ refers only to a pair of boundary points, we use the notation $S|\sigma[t]$ to "extract" from S the subsequence between the boundary points of $\sigma[t]$. In other words, the boundary points of a segmentation can be viewed as operators that when "applied" to a sequence they provide the actual subsequence segments. The set of all k-segmentations, i.e., all possible subsets of $k-1$ boundary points out of the n positions, is denoted by $\mathcal{S}_k$.

For accessing the quality of a segmentation of a sequence S we need a measure of the *variability* of each segment of S. Let $f(S')$ be such a measure defined for all $S' \sqsubseteq S$. We typically assume that f is an easily computable cost function. For example, for real-valued sequences, f is often taken to be the variance of the values in S'. Now, the quality of a k-segmentation $\sigma = \langle \sigma[1], \ldots, \sigma[k]\rangle$ on a sequence S is defined to be

$$f(S|\sigma) = \sum_{t=1}^{k} f(S|\sigma[t]),$$

that is, the sum of the cost function f over all segments of S defined by σ. The *sequence-segmentation* problem is to compute the optimal such k-segmentation

$$\sigma^* = \arg\min_{\sigma \in \mathcal{S}_k} f(S|\sigma).$$

For a wide variety of cost functions, a dynamic programming (e.g., algorithm [3]) can be used for computing the optimal k-segmentation.

In the above formulation, the number of segments k is given in advance. If k is variable, the trivial n-segmentation can typically achieve a zero cost. Thus, a popular way for allowing k to be variable is to add a penalization factor for choosing large values of k, for example, the optimal segmentation is now defined to be

$$\sigma^* = \arg\min_{k, \sigma \in \mathcal{S}_k} f(S|\sigma) + k\gamma.$$

A Bayesian approach for selecting the penalization term to make it proportional to the *description length* [21] of the *segmentation model*. For instance, by assuming that for each segment we need to specify one boundary point and d values (one per dimension), one can choose $\gamma = (d+1)\log(dn)$. An important observation, however, is that the same dynamic programming algorithm with no additional time overhead can be used to compute the optimal segmentation for the variable-k version of the problem. In the following and mainly for clarity of exposition, we will only concentrate on the fixed-k version, however, all of our definitions and algorithms can be applied to the variable-k version, as well.

We now proceed to define the *clustered* version of segmentation problems, which is the focus of our paper. As we explained in the introduction, the main idea is to allow subsets of dimensions to be segmented separately. For the next definition we use the notation $s_i|\sigma$ as we used $S|\sigma$: the segmentation σ is applied to the one-dimensional sequence s_i as it was applied to the multi-dimensional sequence S; the cost function $f(s_i|\sigma)$ is computed accordingly.

PROBLEM 1. *Given a d-dimensional sequence S with n values in each dimension, a cost function f defined on all subsequences and all dimensions of S, and integers k and c, compute c different k-segmentations $\sigma_1, \ldots, \sigma_c$, so as to minimize the sum*

$$\sum_{i=1}^{d} \min_{1 \le j \le c} f(s_i|\sigma_j).$$

In other words, we seek to partition the sequence dimensions in c clusters, and compute the optimal segmentation in each one of those, in a way that the total error is minimized. As before, one can use the Bayesian approach to define the variable-c version of the problem, where the optimal value for c is sought, but again we assume that the value of c is given.

2.1 Connections to related work

The clustered segmentation problem, as stated above, is clearly related with the time-series clustering problem [24, 13]. Several definitions for time-series similarity have been discussed in the literature, for example, see [4, 7, 1]. A key difference, however, is that in our formulation signals are assigned to the same cluster if they can be segmented "well" together, while most time-series clustering algorithms base their grouping criteria in a more geometric notion of similarity. To emphasize the difference, we note that our methods can be used to segment non-numerical sequences like the SNP data we described in the introduction.

The formulation in Problem 1 suggests that one can consider clustered segmentation as a k-median type of problem (e.g., see [18]). However, a main difficulty with trying to apply k-median algorithms in our setting, is that the space of solutions is extremely large. Furthermore, for k-median algorithms it is often the case that a "discretization" of the solution space can be applied (seek for solutions only among the input points). Assuming the triangle inequality, this discretization degrades the quality of the solution by a factor of at most 2. In our setting, however, the solution space (segmentations) is different from the input space (sequence), and also many natural distance functions between sequences and segmentations does not form a metric.

Finally, our problem is also related with the notion of *segmentation problems* as introduced by Kleinberg et al. [16].

In [16], starting from an optimization problem, the "segmented" version of that problem is defined by allowing the input to be partitioned in clusters, and considering the best solution for each cluster separately. To be precise, in the terminology of [16] our problem should be called "segmented segmentation" since in our case the optimization problem is the traditional segmentation problem. Even when starting from very simple optimization problems, their corresponding segmented versions turn out to be hard.

2.2 Problem complexity

In this subsection we demonstrate the hardness of the clustered segmentation problem. In our case, the optimization problem we start with is the traditional non-clustered segmentation problem. This problem is solvable in polynomial time when the k-segmentation variance is considered. Not surprisingly the corresponding clustered segmentation is an NP-hard problem.

THEOREM 1. *Clustered segmentation, as defined in Problem 1, with real-valued sequences, and cost function f the variance function, is NP-hard.*

The proof of the theorem can be found in the Appendix.

3. ALGORITHMS

In this section we describe two classes of algorithms for solving the clustered segmentation problem. In the first class we define distance measures between sequence segmentations and we employ a standard clustering algorithm (e.g., k-means) on the pair-wise distance matrix. The second class consists of two randomized algorithms that cluster sequences using segmentations as "centroids". In particular, we use the notion of a distance between a segmentation and a sequence, which is the error induced to the sequence when the segmentation is applied to it. The algorithms of the second class treat the clustered-segmentation problem as a model selection problem and they try to find the best model that describes the data.

Before proceeding with the description of the algorithms, we briefly review the dynamic-programming algorithm that segments a sequence S into k segments. This algorithm is used by almost all of our methods. The idea of the dynamic programming algorithm is to compute the segmentation incrementally, for all subsequences $S[1,i]$ with $i = 1,\ldots,n$, and all values for an l-segmentation with $l = 1,\ldots,k$. In particular, the computation of an l-segmentation σ for the subsequence $S[1,i]$ is based on the equation

$$f(S[1,i]|\sigma) = \min_{\tau\in\mathcal{S}_{l-1},\, 1\le j<i} f(S[1,j]|\tau) + f(S[j+1,i]).$$

The running time of the dynamic programming algorithm is $O(n^2dkF(n))$, where $F(t)$ is the time required to compute the function f on a subsequence of length t. In the case that f is the variance function, the computation can be done in constant time (by precomputing the sum of values and the sum of squares of values of all prefixes of the sequence), so the running time of the dynamic programming algorithm is $O(n^2dk)$.

3.1 Distance-based clustering of segmentations

In this section we define distance functions between the segmentations of two sequences. Such distance functions can be used to construct a pair-wise distance matrix between the dimensions of the sequence. The distance matrix is then used for clustering the dimensions via a standard distance-based clustering algorithm; in our case, we use standard the k-means algorithm. k-means despite its limitations as a hill-climbing algorithm that is not guaranteed to converge to a global optimum, is mainly used because it is efficient and it works very well in practice. Variations of the k-means algorithm have been proposed for time-series clustering, for example in [24].

The main idea of the k-means algorithm is the following: Given N points that need to be clustered and a distance function d between them, the algorithm starts by selecting k random points as cluster centers and assigning the rest of the $N-k$ points to the closest cluster center, according to d. In that way k clusters are formed. Within each cluster the mean of the points defining the cluster is evaluated and the process continues iteratively with those means as the new cluster centers, until convergence.

The two distance functions defined here are rather intuitive and simple. The first one, D_E, is based on the mutual exchange of optimal segmentations of the two sequences and the evaluation of the additional error such an exchange introduces. Therefore, two sequences are then similar if the best segmentation of the one describes "well" the second, and vice versa. The second distance function, D_P, is probabilistic and it defines the distance between two sequences by comparing the probabilities of each position in the sequence being a segmentation boundary.

3.1.1 Distance as a measure of fit of optimal segmentations

The goal of our clustering is to cluster together dimensions in such a way that similarly segmented dimensions are put in the same cluster, while the overall cost of the clustered segmentation to be minimized. Intuitively this means that a distance function should perform well if it quantifies how well the optimal segmentation of the one sequence describes the other one and vice versa. Based on exactly this notion of "exchange" of optimal segmentations of sequences, we define the distance function D_E in the following way.

Given two dimensions s_i, s_j and their corresponding optimal k-segmentations $\sigma_i^*, \sigma_j^* \in \mathcal{S}_k$ we define the distance of s_i from σ_j^* denoted by $D_E(s_i, \sigma_i^*|\sigma_j^*)$ as follows:

$$D_E(s_i, \sigma_i^*|\sigma_j^*) = f(s_i|\sigma_j^*) - f(s_i|\sigma_i^*)$$

However in order for the distance between two sequences and their corresponding segmentations to be symmetric we alternatively use the following symmetric definition of D_E:

$$D_E(s_i, \sigma_i^*, s_j, \sigma_j^*) = D_E(s_i, \sigma_i^*|\sigma_j^*) + D_E(s_j, \sigma_j^*|\sigma_i^*)$$

3.1.2 Probabilistic distance

Distance function D_P is based on comparing two dimensions by comparing the probability distributions of their

points being segment boundaries.[1] For each dimension s_i with $1 \leq i \leq d$ we associate a probability distribution p_i. The value of $p_i[t]$ at point t with $1 \leq t \leq n$ corresponds to the probability of the t-th point of the series being a segment boundary. Associating a probability distribution with each sequence, allows us to define the distance between two sequences as the variational distance between the corresponding distributions. Therefore, we define the distance function $D_P(s_i, s_j)$ between the dimensions s_i and s_j as follows:

$$D_P(s_i, s_j) = \mathrm{Var}(p_i, p_j) = \sum_{1 \leq t \leq n} |p_i[t] - p_j[t]| \qquad (1)$$

Computing the probabilities of segment boundaries for a given sequence and given the required k number of segments, can be done in $O(n^2k)$ time using dynamic programming. Consider a dimension s of our d-dimensional sequence. Let the probability that there is a segment boundary at point t of s, when segmented using k-segments. Denote by $\mathcal{S}_k^{(t)}$ the set of all k-segmentations having a boundary at point t. Then we are interested in the probability of any segmentation from the set $\mathcal{S}_k^{(t)}$ given the sequence s:

$$p(\mathcal{S}_k^{(t)}|s) = \sum_{\sigma \in \mathcal{S}_k^{(t)}} p(\sigma|s).$$

Since $\mathcal{S}_k$ refers to the set of all k-segmentations of the full sequence s, the above equation can be rewritten as follows:

$$p(\mathcal{S}_k^{(t)}|s) = \frac{\sum_{\sigma' \in \mathcal{S}_k^{(t)}} p(\sigma', s)}{\sum_{\sigma \in \mathcal{S}_k} p(\sigma, s)}$$

For the joint probabilities of segmentation and sequence, $p(\sigma, s)$, it holds that:

$$p(\sigma, s) = \frac{1}{Z} 2^{-\sum_{[a,b] \in \sigma} f(s[a,b]|\sigma)}.$$

In the above equation Z is a normalizing constant that cancels out. For building the dynamic programming equations we need to define the following entity for a segmentation σ:

$$q(a, b) = 2^{-f(s[a,b]|\sigma)}.$$

Additionally, for any interval $[t, t']$ and considering segmentations consisting of i-segments (where $1 \leq i \leq k$) we define:

$$Q_i(t, t') = \sum_{\sigma \in \mathcal{S}_i[t,t']} \prod_{[a,b] \in \sigma} q(a, b).$$

Since $\mathcal{S}_k[t, t+1]$ is equal to the Cartesian product $\mathcal{S}_i[1, t] \times \mathcal{S}_{k-i}[t+1, n]$ for $1 \leq i \leq k$ we have that:

$$p(\mathcal{S}_k^{(t)}|s) = \sum_{1 \leq i \leq k} \frac{Q_i(1, t) Q_{k-i}(t+1, n)}{Q_k(1, n)}.$$

Finally the inner-most equations for the dynamic programming, that consider segmentations of a fixed number of segments i (with $1 \leq i \leq k$) are:

$$Q_i(1, b) = \sum_{1 \leq a \leq b} Q_{i-1}(1, a-1) q(a, b)$$

and

$$Q_i(a, n) = \sum_{a \leq b \leq n} q(a, b) Q_{i-1}(b+1, n).$$

[1] For a similar development see [17].

Using the above equations we can compute the probabilities of each point being a segment boundary in each one of the d dimensions of S. Pairwise distances between two dimensions are evaluated using Equation (1).

3.2 Non-distance based clustering of segmentations

In this section we describe two algorithms that treat clustered segmentation as a model-selection problem. The first algorithm, SAMPLSEGM, is a sampling algorithm and it is motivated by the theoretical work of Kleinberg et al. [16], Indyk [11, 12] and Charikar et al. [5]. The second, ITERCLUSTSEGM, is an adaptation of the popular k-means algorithm. Both algorithms are simple and intuitive and they perform well in practice.

3.2.1 The SAMPLSEGM algorithm

The basic idea behind the SAMPLSEGM approach is the intuition that if the data exhibit clustered structure, then a small sample of the data would exhibit the same structure. The reason is that for large clusters in the data set one would expect to sample enough data, so that similar clusters appear in the sampled data. On the other hand, one can possibly afford to miss data from small clusters in the sampling process, because small clusters do not contribute much in the overall error function. Our algorithm is motivated by the work of Kleinberg et al. [16], in which a sampling algorithm for the segmented version of the catalog problem is proposed. Similar ideas have been used successfully by Indyk [11, 12] for the problem of clustering in metric spaces.

For the clustered segmentation problem we adopt a natural sampling-based technique: We first sample uniformly at random a small set A of $r \log d$ dimensions, where r is a small constant. Then we search exhaustively all possible *partitions* of A into c clusters $A_1, \ldots, A_c$. For each cluster A_j we find the optimal segmentation $\sigma_j \in \mathcal{S}_k$ for the sequence S on the dimensions that are associated with A_j. The rest of the dimensions s_i that are not included in the sample, are assigned to the set j that minimizes the error $f(s_i|\sigma_j)$. The partition of the sample set A that causes the least error is considered to be the solution found for the set A. The whole sampling process is repeated with different sample sets A for a small number of times (in our experiments 3 times) and the best result is reported as the output of the sampling algorithm.

When the size of the sample set is logarithmic in the number of dimensions, the overall running time of the algorithm is polynomial. In our experiments, we found that the method is accurate for data sets of moderate size, but it does not scale well for larger data sets.

3.2.2 The ITERCLUSTSEGM algorithm

The ITERCLUSTSEGM algorithm is an adaptation of the widely-used k-means algorithm where the cluster means are replaced by the common segmentation of the dimensions in the cluster and the distances of a sequence to the cluster is the error induced when the cluster's segmentation is applied to the sequence.

Therefore in our case, the c centers correspond to c different

segmentations. The algorithm is iterative and at the t-th iteration step it keeps an estimate for the solution segmentations $\sigma_1^t, \ldots, \sigma_c^t$, which is to be refined in the consecutive steps. The algorithm starts with a random clustering of the dimensions, and it computes the optimal k-segmentation for each cluster. At the $(t+1)$-th iteration step, each dimension s_i is assigned to the segmentation σ_j^t for which the error $f(s_i|\sigma_j^t)$ is minimized. Based on the newly obtained clusters of dimensions, new segmentations $\sigma_1^{t+1}, \ldots, \sigma_c^{t+1}$ are computed, and the process continues until there is no more improvement in the error. The complexity of the algorithm is $O(I(cd+cP(n,d)))$, where I is the number of iterations until convergence, and $P(n,d)$ is the complexity of segmenting a sequence of length n and d dimensions.

4. EXPERIMENTS

In this section we describe the experiments we performed in order to evaluate the validity of the clustered segmentation model and the behavior of the suggested algorithms. For our experiments we used both synthetically generated data, as well as real data consisting of time series and genomic sequences. For the synthetic data we report that in all cases the true underlying model, used to generate the data, is found. For the real data we found that in all cases clustered segmentations, output by the proposed algorithms, produce better models than the models produced by non-clustered segmentations.

4.1 Ensuring fairness in model comparisons

In the experimental results shown in this section we report the accuracy in terms of errors. Our intention is to consider the error as a measure of comparing models: a smaller error indicates a better model. However, this is the case when the compared models have the same number of parameters. It would be unfair to compare the errors induced by two models with different number of parameters, because in this case the trivial model of each point described by itself would induce the the least error and would be the best.

Therefore, to make the comparison between two different models fair, we are taking care to ensure that the same number of parameters is used in both models. We briefly describe the methodology we followed in order to guarantee fairness in comparing the different models. Consider a k-segmentation for a d-dimensional sequence S of length n. If no clustering of dimensions is considered, the number of parameters that are necessary to describe this k-segmentation model is $k(d+1)$. This number comes from the fact that we can describe the model by specifying the starting point and the d mean values—one for each dimension—for each one of the k segments. Consider now a clustered segmentation of the sequence with c clusters and k' segments for each cluster. The number of parameters for this model is $d + \sum_{i=1}^{c} k'(d_i+1) = d + k'(d+c)$, since, in addition to specifying the starting points and the values for each cluster, we also need d parameters to indicate the cluster that each dimension belongs to. In our experiments, in order to compare the errors induced by the two models we select parameters so that $k(d+1) = d + k'(d+c)$.

4.2 Experiments on synthetic data

We first describe our experiments on synthetic data. For the purpose of this experiment, we have generated sequence data from a known model, and the task is to test if the suggested algorithms are able to discover that model.

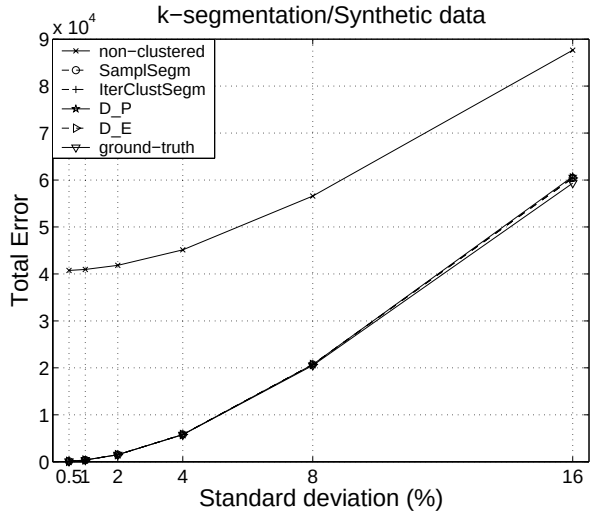
The data we used were generated as follows: the d dimensions of the generated sequence were divided in advance into c clusters. For each cluster we select k segment boundaries, which are common for all the dimensions in that cluster, and for the j-th segment of the i-th dimension we select a mean value μ_{ij}, which is uniformly distributed in $[0,1]$. Points are then generated by adding a noise value sampled from the normal distribution $\mathcal{N}(\mu_{ij}, \sigma^2)$. An example of a small data set generated by this method is shown in Figure 1. For our experiments we fixed the values $n = 1000$ points, $k = 10$ segments, and $d = 200$ dimensions. We created different data sets using $c = 2, ..., 6$ clusters and with standard deviations varying from 0.005 to 0.16.

The results for the synthetically generated data are shown in Figure 2. One can see that the errors of the reported clustered segmentation models are typically very low for all of our algorithms. In most of the cases all proposed methods approach the *true* error value. Here we report the results for small sample sizes (usually $(c+4)$ samples with c being the number of clusters). Since our algorithms are randomized we repeat each one of them for 5 times and report the best found solution. Apart from the errors induced by the proposed algorithms, the figures include also two additional errors. The error induced by the non-clustered segmentation model with the same number of parameters and the error induced by the true model that has been used for generating the data ("ground-truth"). The first one is always much larger than the error induced by the models reported by our algorithms. In all the comparisons between the different segmentation models we take into consideration the fairness criterion discussed in the previous subsection.

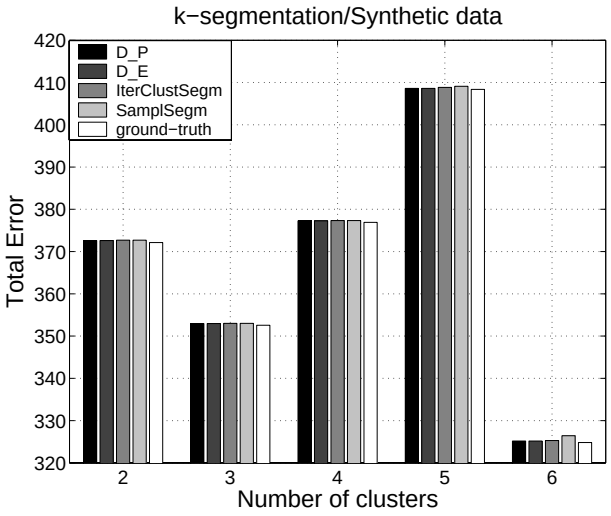
As indicated in Figure 2(a) the difference in errors becomes smaller as the standard deviation increases. This is natural since as standard deviation increases all dimensions tend to become uniform and the segment structure disappears. The error of the clustered segmentation model for different number of clusters is shown in Figure 2(b). The better performance of the clustered model is apparent. Notice that the error caused by the non-clustered segmentation is an order of magnitude larger than the corresponding clustered segmentation results and thus omitted from the plot.

4.3 Experiments on time-series data

Next, we tested the behavior of the clustered segmentation model on real time-series data sets obtained by the UCR time-series data mining archive [15]. We used the **phone** and the **spot_exrates** data sets of the archive. The **phone** data set consists of 8 dimensions each one corresponding to the value of a sensor attached to a mobile phone. For the clustered segmentation we used number of segments $k = 8$ and number of clusters $c = 2$, 3 and 4, while for the non-clustered segmentation we used $k = 10$, 11, and 11, respectively so that we again guarantee fair comparison. Figure 3(a) shows the error induced by the clustered and the non-clustered segmentations for different number of clusters.

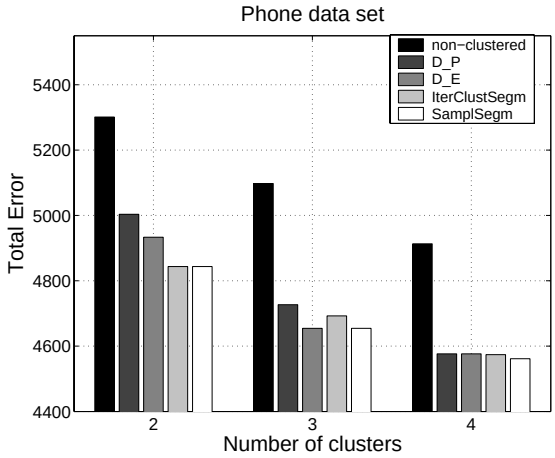


(a) Total error of the model as a function of standard deviation in the generated data.

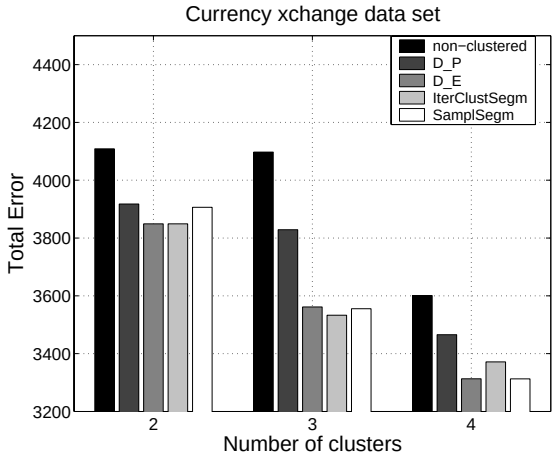


(b) Total error as a function of the number of clusters existing in the generated data. The bar that corresponds to the non-clustered model is missing since it is an order of magnitude larger.

Figure 2: Error for k-segmentations on synthetic data sets.



(a)



(b)

Figure 3: Experiments with real time-series data.

Apparently, the clustered segmentation model reduces by far the induced error. SAMPLSEGM, ITERCLUSTSEGM and clustering using D_E are all giving the same level of error, with clustering using D_P performing almost equally well, and in all cases better than the non-clustered segmentation.

Analogous results are obtained for the **spot_exrates** data set as illustrated in Figure 3(b). This data set contains the spot prices (foreign currency in dollars) and the returns for daily exchange rates of the 12 different currencies relative to the US dollar. There are 2567 (work-)daily spot prices, and so 2566 daily returns for each of these 12 currencies, over a period of about 10 years (10/9/86 to 8/9/96). Again, we considered the case of k-segmentations. For the clustered segmentation we used number of segments $k = 10$ and number of clusters $c = 2$, 3, and 4, while for the non-clustered segmentation we used $k = 12$, 12 and 13, respectively.

4.4 Experiments on mobile-sensor data

Finally, we tested the behavior of the proposed methods on the benchmark dataset for context recognition described in [19].[2] The data were recorded using microphones and a sensor box, attached to a mobile phone. The combination was carried by the users during the experiments and the data were logged by a laptop carried by the users. The signals collected were transformed into 29 variables the values of which have been recorded for some periods of time.

The dataset basically contains 5 scenarios each one repeated for a certain number of times. The 29 variables recorded, that correspond to the 29 dimensions, are related to *device position*, *device stability*, *device placement*, *light*, *temperature*, *humidity*, *sound level* and *user movement*.

The results of the clustered segmentations algorithms for scenario 1 and 2 are shown in Figures 4 and 5. For the rest of the scenarios the results are similar and thus omitted. Since some dimensions in the different scenarios are all constant we have decided to ignore them. Therefore from a total of 29 dimensions we have considered 20 for scenario 1, 19 for scenario 2, 16 for scenario 3, 14 for scenario 4 and 15 for scenario 5.

For the case of clustered segmentation we considered $k = 5$ and $c = 2, 3, 4, 5$ for all the scenarios, while the corresponding values of k for the non-clustered segmentation that could guarantee fairness of comparison of the results was evaluated to be $k = 6, 7$ depending on the value of c and the number of dimensions in the scenario. The error levels using the different methods proposed here, as well as the non-clustered segmentation algorithm are shown in Figures 4 and 5. In all cases the clustered segmentation model found by any of our four methods has much lower error level than the corresponding non-clustered one. Some indicative clusterings of the dimensions of scenario 1 using the proposed methods are shown in Figure 6. Notice that the clustering shown in Figure 6 reflects an expected clustering of dimensions. The first cluster contains only dimensions related to the "Position", the "Stability" and the "Placement" of the device. The second cluster puts together all time series related to the "Humidity" of the environment. The third cluster consists of dimensions related to the "Light" and the "Sound Pressure" of the environment as well as the user movement. Finally the last cluster contains all the dimensions related to the "Temperature" of the environment as well as the dimensions that corresponds to the "Running" dimensions that characterizes the user movement. Although this last results raises some suspicious, since running is the only user action related dimension that is clustered separately from the other two, it can be quite easily explained if we observe Figure 8. It is obvious that segmentation-wise dimensions "UserAction:Movement:Walking" and "'UserAction:Movement:WalkingFast" are much closer to each other than they are with "UserAction:Movement:Running", while the latter one can be easily segmented using segmentation boundaries of dimensions "Environment:Temperature:Warm" and "Environment:Temperature:Cool". Similar observations can be made also for the rest of the clusterings obtained using the other three proposed methods. Indicatively we show

[2]The dataset is available at
http://www.cis.hut.fi/jhimberg/contextdata/index.shtml

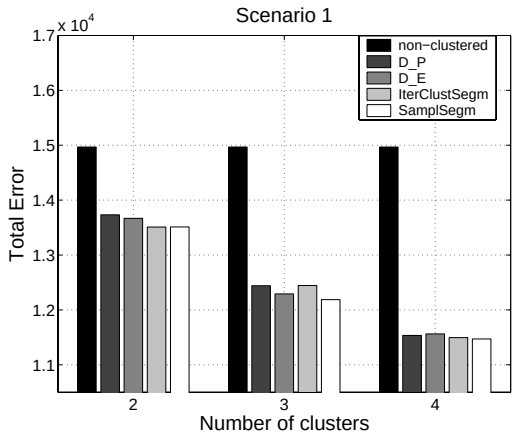


Figure 4: Experiments with scenario 1.

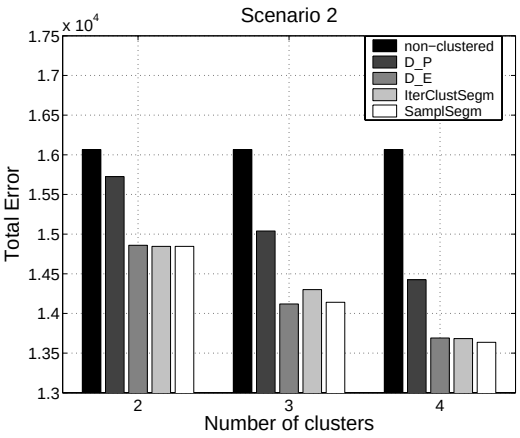


Figure 5: Experiments with scenario 2.

in Figure 7 the clustering obtained using k-means algorithm for the same number of clusters and using L_1 as the distance metric between the different dimensions. There is not an obvious correspondence between the clustering found in that case and the expected clustering with respect to the general categories each dimension belongs to.

4.5 Discussion

The experimental evaluation performed on both synthetic and real data indicates that the clustered segmentation model is a more precise alternative for describing the data at hand, and all the proposed methods find models that show much smaller error than the error of the equivalent non-clustered segmentation model, in all the considered cases. Overall, in the case of synthetic data the true underlying model, used for the data generation, is always found. For the real data, the true model is unknown and thus we base our conclusions on the errors induced by the two alternative models when the same number of parameters is used.

The proposed algorithms are intuitive and they perform well in practice. For most of the cases they give equivalent results and they find almost the same models. Noticeably, SamplSegm appears to be competitive in terms of quality of

Device:Position:DisplayDown, Device:Position:AntennaUp, Device:Stability:Stable, Device:Stability:Unstable, Device:Placement:AtHand
Environment:Humidity:Humid, Environment:Humidity:Normal, Environment:Humidity:Dry
Environment:Light:EU, Environment:Light:Bright, Environment:Light:Normal, Environment:Light:Dark, Environment:Light:Natural, Environment:SoundPressure:Silent, Environment:SoundPressure:Modest, UserAction:Movement:Walking, UserAction:Movement:WalkingFast
Environment:Temperature:Warm, Environment:Temperature:Cool, UserAction:Movement:Running

Figure 6: Clustering of the dimensions of Scenario 1 into 4 clusters using ITERCLUSTSEGM algorithm.

Environment:Humidity:Dry
Environment:Light:Bright, Environment:Light:Natural, UserAction:Movement:WalkingFast
Device:Position:AntennaUp, Device:Stability:Unstable, Environment:Light:EU, Environment:Light:Normal, Environment:Temperature:Warm, Environment:Humidity:Humid, Environment:SoundPressure:Silent, 'UserAction:Movement:Walking
Device:Position:DisplayDown, Device:Stability:Stable, Device:Placement:AtHand, Environment:Light:Dark, Environment:Temperature:Cool, Environment:Humidity:Normal, Environment:SoundPressure:Modest, UserAction:Movement:Running'

Figure 7: Clustering of the dimensions of Scenario 1 into 4 clusters using L_1 distance k-means.

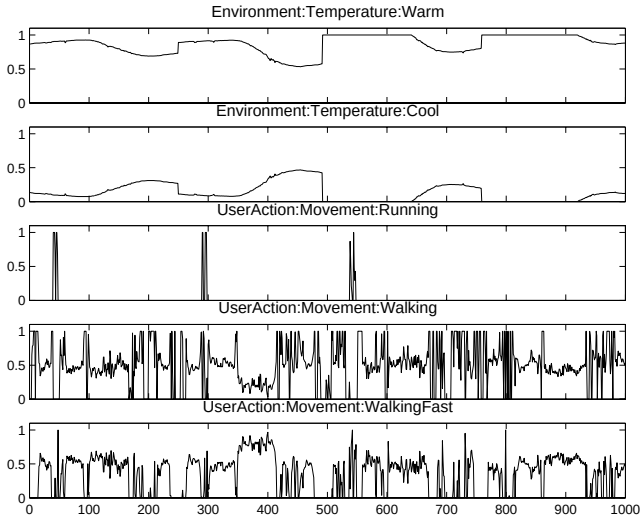


Figure 8: Subset of dimensions of Scenario 1 dataset that explain clustering in Figure 6.

results even for small sample sizes. However, it can become slow as the sample size increases since we need to examine all possible partitions of the sample and select the one that implies the best model. On the other hand, the other three alternatives scale smoothly and appears to perform well in practical settings.

5. CONCLUSIONS

We have introduced the clustered segmentation problem where the task is to cluster the dimensions of a multidimensional sequence into c clusters so that the dimensions grouped in the same cluster share the same segmentation points. The problem when considered for real-valued sequences and cost function the variance function is NP-hard. We described simple algorithms for solving the problem. All proposed methods perform well for both synthetic and real data sets consisting of time-series data. In all the cases we experimented with, the clustered segmentation model seems to describe the datasets better than the corresponding non-clustered segmentation model with the same number of parameters.

6. REFERENCES

[1] R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceedings of the 21st International Conference on Very Large Data Bases*, pages 490–501, Zurich, Switzerland, 1995.

[2] R. K. Azad, J. S. Rao, W. Li, and R. Ramaswamy. Simplifying the mosaic description of DNA sequences. *Physical Review E, 66, article 031913*, 2002.

[3] R. Bellman. On the approximation of curves by line segments using dynamic programming. *Communications of the ACM*, 4(6), 1961.

[4] B. Bollobas, G. Das, D. Gunopulos, and H. Mannila. Time-series similarity problems and well-separated

geometric sets. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 454–456, 1997.

[5] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 30–39, 2003.

[6] M. J. Daly, J. D. Rioux, S. F. Schaffner, T. J. Hudson, and E. S. Lander. High-resolution haplotype structure in the human genome. *Nature Genetics*, 29:229–232, 2001.

[7] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time series databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 419–429, 1994.

[8] A. Gionis and H. Mannila. Finding recurrent sources in sequences. In *Proceedings of the 7th Annual International Conference on Research in Computational Molecular Biology*, pages 115–122, Berlin, Germany, 2003.

[9] D. Gusfield. Haplotyping as perfect phylogeny: conceptual framework and efficient solutions. In *International Conference on Research in Computational Molecular Biology*, pages 166–175, 2002.

[10] J. Himberg, K. Korpiaho, H. Mannila, J. Tikanmäki, and H. Toivonen. Time series segmentation for context recognition in mobile devices. In *Proceedings of the IEEE International Conference on Data Mining*, pages 203–210, 2001.

[11] P. Indyk. Sublinear time algorithms for metric space problems. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 428–434, 1999.

[12] P. Indyk. A sublinear time approximation scheme for clustering in metric spaces. In *Proceedings of the Symposium on Foundations of Computer Science*, pages 154–159, 1999.

[13] K. Kalpakis, D. Gada, and V. Puttagunta. Distance measures for effective clustering of arima time-series. In *Proceedings of the IEEE International Conference on Data Mining*, pages 273–280, 2001.

[14] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *Proceedings of the IEEE International Conference on Data Mining*, pages 289–296, 2001.

[15] E. Keogh and T. Folias. The UCR time series data mining archive, 2002.

[16] J. Kleinberg, C. Papadimitriou, and P. Raghavan. Segmentation problems. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 473–482, 1998.

[17] M. Koivisto, M. Perola, T. Varilo, et al. An MDL method for finding haplotype blocks and for estimating the strength of haplotype block boundaries. In *Pacific Symposium on Biocomputing*, pages 502–513, 2003.

[18] J.-H. Lin and J. S. Vitter. ϵ-approximations with minimum packing constraint violation. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 771–782, 1992.

[19] J. Mantyjarvi, J. Himberg, P. Kangas, U. Tuomela, and P. Huuskonen. Sensor signal data set for exploring context recognition of mobile devices. In *In Workshop Benchmarks and a database for context recognition in conjuction with the 2nd Int. Conf. on Pervasive Computing (PERVASIVE 2004)*, 2004.

[20] N. Patil, A. J. Berno, D. A. Hinds, et al. Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21. *Science*, 294:1669–70, 2001.

[21] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[22] M. Salmenkivi, J. Kere, and H. Mannila. Genome segmentation using piecewise constant intensity models and reversible jump MCMC. In *Proceedings of the European Conference on Computational Biology*, pages 211–218, 2002.

[23] R. Schwartz, B. V. Halldorsson, V. Bafna, A. G. Clark, and S. Istrail. Robustness of inference of haplotype block structure. *Journal of Computational Biology*, 10(1):13–9, 2003.

[24] M. Vlachos, J. Lin, E. Keogh, and D. Gunopulos. A wavelet-based anytime algorithm for k-means clustering of time series, 2003.

[25] J. D. Wall and J. K. Pritchard. Assessing the performance of the haplotype block model of linkage disequilibrium. *American Journal of Human Genetics*, 73:502–515, 2003.

Appendix

THEOREM 1. *Clustered segmentation, as defined in Problem 1, with real-valued sequences, and cost function f the variance function, is NP-hard.*

PROOF. We give only an outline of the proof idea. The problem from which we obtain the reduction is the SET COVER, a well-known NP-hard problem. An instance of the SET COVER specifies a ground set U of n elements, a collection $\mathcal{C}$ of m subsets of U, and a number c. The question is whether there are c sets in the collection $\mathcal{C}$ whose union is the ground set U.

Given an instance of the SET COVER, we create an instance of the clustered k-segmentation as follows: We form a sequence with n dimensions. In each dimension there are $2m + 1$ "runs" of values and each run has an equal number of values. There are two types of runs: `High` and `Low`, and these two types are alternating across the sequence. All `High` runs have all their values equal to 1. The `Low` runs are indexed from 1 to m and they in turn can be of two types: Z and E. Z-`Low` runs have all their values equal to 0. E-`Low` runs are split in two pieces, so that the f function on such a run incurs

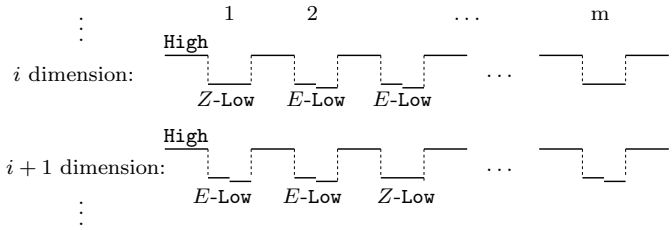


Figure 9: The construction used in the proof of Theorem 1.

cost exactly ϵ. The construction can be seen schematically in Figure 9.

The instance of the SET COVER is encoded in the `Low` runs. If the j-th set of $\mathcal{C}$ contains the i-th element of U, then the j-th `Low` run of the i-th dimension is set to type E, otherwise it is set to type Z. Assume that in total there are L `Low` runs of type E. We would ask for a k-segmentation with $k = 2m + 2$ segments. By setting ϵ to be very small, the $2m + 1$ segments would be separating the `High` from the `Low` runs, and there would be the freedom to segment one more E-`Low`-type run in order to save an additional cost of ϵ. The question to ask is if there is a clustered segmentation with c clusters that has cost at most $(L - n)\epsilon$. One can show that is possible if and only if there is a solution to the original SET COVER problem. Furthermore, ϵ needs only to be polynomial in $1/n$ and $1/m$, so the transformation can be computed in polynomial time. □

Fast algorithms for frequent episode discovery in event sequences

Srivatsan Laxman
Dept. of Electrical Engineering
Indian Institute of Science
Bangalore, India
Email: srivats@ee.iisc.ernet.in

P. S. Sastry
Dept. of Electrical Engineering
Indian Institute of Science
Bangalore, India
Email: sastry@ee.iisc.ernet.in

K. P. Unnikrishnan
General Motors R&D Center
Warren, MI, USA
Email: unni@gmr.com

Abstract

In this paper we consider the process of discovering frequent episodes in event sequences. The most computationally intensive part of this process is that of counting the frequencies of a set of candidate episodes. We present two new frequency counting algorithms for speeding up this part. These, referred to as non-overlapping and non-inteleaved frequency counts, are based on directly counting suitable subsets of the occurrences of an episode. Hence they are different from the frequency counts of Mannila et al [1], where they count the number of windows in which the episode occurs. Our new frequency counts offer a speed-up factor of 7 or more on real and synthetic datasets. We also show how the new frequency counts can be used when the events in episodes have time-durations as well.

1. INTRODUCTION

Development of data mining techniques for time series data is an important problem of current interest [1, 2, 3, 4, 5]. An interesting framework for temporal data-mining in the form of discovering frequent episodes in event sequences was first proposed in [1]. This framework has been found useful for analyzing, e. g. alarm streams in telecommunication networks, logs on web servers, etc. [1, 6, 7]. Further, in [2], the formalism for episode discovery was extended to incorporate time durations into the episode definitions. For many data sets, such as the manufacturing plant data analyzed in this paper, this extended framework provides a richer and more expressive class of patterns in the temporal datamining context. This paper describes two new algorithms useful in the general framework of frequent episode discovery.

This paper proposes two alternative measures for frequency of an episode. Both these are more closely related to the number of occurrences of episodes than the windows-based frequency count proposed in [1]. We show through some empirical investigations that our method of counting would also result in essentially the same kind of episodes being discovered as frequent. However, our frequency counting algorithms need less temporary memory and more importantly they speed up the process of discovering frequent episodes by a factor of 7 (or more) thus rendering the framework of episode discovery attractive in many more applications.

The following is a quick introduction to the framework proposed in [1]. The data is a sequence of events given by $\langle (E_1, t_1), (E_2, t_2), \ldots \rangle$ where E_i represents an *event type* and t_i the corresponding *time of occurrence* of the event. All E_i belong to a finite set of event types. For example, the following is an event sequence containing seven events:

$$\langle (A,1), (B,3), (C,4), (E,12), (A,14), (B,15), (C,16) \rangle. \quad (1)$$

Note that in all our examples, we use the set $\{A, B, C, \ldots\}$ as the set of event types. An episode is an ordered tuple of event types[1]. For example, $(A \rightarrow B \rightarrow C)$ is a 3-node episode. An episode is said to *occur* in an event sequence if we can find events in the sequence which have the same time ordering as that specified by the episode. In the example sequence given by (1), an occurrence of the episode $(A \rightarrow B \rightarrow C)$ is constituted by the events $(A,1), (B,3), (C,4)$. Note that the events $(A,14), (B,3), (C,4)$ do not constitute an occurrence because the episode demands that A precedes B and C. Another occurrence of the episode here is $(A,1), (B,3), (C,16)$. It is easy to observe that there are altogether four occurrences of this episode in the data sequence given by (1). A *subepisode* is a subsequence of the episode which has the same ordering as the episode. For example, $(A \rightarrow B)$, $(A \rightarrow C)$ and $(B \rightarrow C)$ are the 2-node subepisodes of the 3-node episode $(A \rightarrow B \rightarrow C)$, while $(B \rightarrow A)$ is not. The frequency of an episode can be defined in many ways. However, for a frequency count to be reasonable, its definition must guarantee that any subepisode is at

[1] In the formalism of [1], this corresponds to the *serial episode*.

least as frequent as the episode.

A frequent episode is one whose frequency exceeds a user specified threshold. The procedure for discovering frequent episodes proposed in [1] is based on the same general idea as the Apriori algorithm [5]. First we compute all frequent 1-node episodes. Then these are combined in all possible ways to make candidate 2-node episodes. By calculating the frequencies of these candidates, obtain all frequent 2-node episodes. These are then used to obtain candidate 3-node episodes and so on. In our method we use the same procedure for generating candidate episodes as in [1].

2. FREQUENCY COUNTING

The main computationally intensive step in frequent episode discovery is that of calculating the frequency of sets of candidate episodes. This needs to be achieved using as few database passes as possible. In [1], Mannila, et. al., suggested a frequency count which is defined as the number of (fixed width) windows (over the data) in which the episode (whose frequency we are counting) occurs at least once. Multiple occurrences of the episode in the same window has no effect on this frequency measure. Since episodes are essentially temporal patterns their occurrence(s) can be recognized using finite state automata. For the windows-based count, Mannila, et. al. [1] present a fairly efficient algorithm that uses n automata per episode to count frequencies of n-node episodes.

The windows-based frequency count however, is not easily relatable to the general notion of frequency, namely, the number of occurrences of an episode in the data. Further, it is sensitive to a user-specified window width. Another windows-based frequency count has been recently proposed [8] in which the size of the window grows automatically to accommodate episode occurrences that are more spread out. In [1], Mannila et al. proposed an alternative frequency measure which is defined as the number of *minimal* occurrences of an episode. A minimal occurrence is essentially a *window* in which the episode occurs such that no proper sub-window of it contains an occurrence of the episode. However, the informal algorithm suggested in [1] for counting this frequency is very inefficient in terms of the temporary memory needed. (The space complexity of this method is of the order of the length of the data sequence).

Intuitively, the number of occurrences of an episode seems to be the most natural choice for its frequency. As proposed in [1], the occurrence of an episode in an event sequence may be recognized by using a finite state automaton that accepts the episode and rejects all other input. For example, for the episode $(A \rightarrow B \rightarrow C)$, we would have a automaton that transits to state 'A' on seeing an event of type A and then waits for an event of type B to transit to its next state and so on. When this automaton transits to its final state we have recognized an occurrence of the episode. We need different instances of the automaton of an episode to keep track of all its state transition possibilities and hence count all its occurrences. If we want to count *all* occurrences, the number of automata can become unbounded. Consider again the example given earlier. When we see the event $(A, 1)$ we can transit an automaton of this episode into state A. When we see the next event $(B, 3)$, we cannot simply let this automaton transit to state B. That way, we would miss an occurrence which uses the event $(A, 1)$ but some other occurrence of the event type B. Hence, when we see $(B, 3)$, we need to keep one automaton in state A and transit another instance of automaton for this episode into state B. As is easy to see, we may need spawning of arbitrary number of new instances of automata if we do not want to miss counting any occurrence. Hence, the question now is how should we suitably restrict the class of occurrences so that we can find an efficient counting procedure.

Each occurrence of an episode is associated with a set of events in the data stream. We say two occurrences are distinct if they do not share any events. In the data sequence given by (1), there are only two distinct occurrences of $(A \rightarrow B \rightarrow C)$ though the total number of occurrences are four. It may appear that if we restrict the count to only distinct occurrences, then we do not need to spawn unbounded number of automata. However, this is not true, as can be seen from the following example. Consider the sequence

$$\langle (A, 1), (B, 2), (A, 3), (B, 4), (A, 7), (B, 8), \ldots \rangle. \qquad (2)$$

In such a case, we may need (in principle) any number of instances of the $(A \rightarrow B \rightarrow C)$ automaton, all waiting in state 2, since the there may be any number of occurrences of the event type C later in the event sequence.

Hence there is a need for further restricting the kinds of occurrences to count when defining the frequency. We define two such frequencies below, one based on what we call *non-overlapping* occurrences and the other based on what is termed *non-interleaved* occurrences.

Two occurrences of an episode in an event sequence are called *non-overlapping* if any event corresponding to one occurrence does not happen to be in between events corresponding to the other occurrence. In (1) there are only two non-overlapping occurrences of $(A \rightarrow B \rightarrow C)$. In (2) we need to keep track of only the last pair of event types A and B if all we need to recognize are non-overlapping occurrences $(A \rightarrow B \rightarrow C)$. We note in passing here that this frequency count based on non-overlapping episode occurrences is also interesting because it facilitates a formal connection between frequent episode discovery and HMM learning [9].

Although, the idea of counting only non-overlapping occurrences is theoretically elegant and practically attractive, it does appear to constrain the occurrence possibilities for an episode in the event sequence. There is thus a need to find a way to count at least some overlapping occurrences as well. Towards this end we define *non-interleaved* occurrences of an episode.

Each occurrence of an episode is a 1-1 map from the nodes of the episode to events in the data sequence. For an episode α, we denote the number of nodes in it by $|\alpha|$ and the ordered sequence of nodes of α is referred to as $v_1, v_2, \ldots$. Let h_1 and h_2 denote two different occurrences of α. Thus, $h_1(v_i)$ denotes the event in the data sequence that corresponds to the node v_i of the episode in the occurrence represented by h_1. By $h_1(v_i) < h_2(v_j)$, we mean that the event (in the data) corresponding to node v_i in occurrence h_1 has an

earlier occurrence time than that of the event corresponding to node v_j in occurrence h_2. Two occurrences, h_1 and h_2, of an episode α are said to be *non-interleaved* if either

$$h_2(v_j) > h_1(v_{j+1}) \ \forall j, 1 \leq j < |\alpha|,$$

or

$$h_1(v_j) > h_2(v_{j+1}) \ \forall j, 1 \leq j < |\alpha|.$$

We illustrate the definition with an example. Consider an event sequence

$$\langle (A,1),(B,3),(A,4),(A,7),(E,12),(B,14), \\ (B,15),(C,16),(C,18),(B,19),(C,23)\rangle. \quad (3)$$

Consider the same episode as earlier, namely, $\alpha = (A \rightarrow B \rightarrow C)$. Here, there are three distinct occurrences corresponding to: $(A,1),(B,3),(C,16)$, $(A,4),(B,14),(C,18)$ and $(A,7),(B,15),(C,23)$. Since there are only three occurrences of the event type A, we cannot have more than three distinct occurrences. Since any two occurrences are overlapping, there can be at most one non-overlapping occurrences of α. When counting *non-interleaved* occurrences, there can be a maximum of two occurrences: $(A,1),(B,3),(C,16)$ and $(A,4),(B,19),(C,23)$ (or alternatively, $(A,1),(B,3),(C,18)$ and $(A,4),(B,19),(C,23)$).

In terms of the automata that recognize each occurrence, this definition (of non-interleaved) means the following. An instance of the the automaton for α can transit into the state corresponding to a node, say v_2, only if an earlier instance (if any) of the automaton has already transited into state v_3 or higher. Non-interleaved occurrences would include some overlapped occurrences though they do not include all occurrences.

We present two algorithms (Algorithm A and B respectively) for counting non-overlapping and non-interleaved occurrences. Both these use as many automata per episode as there are states in it which is same as the number needed for the window-based frequency count of [1]. Given a set of candidate episodes $\mathcal{C}$, the algorithms return the set of frequent episodes $\mathcal{F}$. Each frequent episode has frequency above a user specified threshold (which is an input to the algorithm). The other input is the data represented as $\mathbf{s} = (s, T_s, T_e)$ where s is the sequence of events with T_s and T_e being the start and end times of that sequence.

2.1 Non-overlapping occurrences

Algorithm A counts the number of non-overlapping occurrences. At the heart of this automata-based counting scheme is the $waits(\cdot)$ list. Since there are many candidate episodes and for each of which there are multiple occurrences, at any time there would be many automata waiting for many event types to occur. In order to traverse and access the automata efficiently, for each event type A, the automata that accept A are linked together in the list $waits(A)$. The $waits(A)$ list contains entries of the form (α, j) meaning that an automaton of the episode α is waiting for event type A as its j^{th} event. That is, if the event type A occurs now in the event sequence, this automaton would accept it and transit to the j^{th} state. At any time (including at the start of the counting process) there would be automata waiting for the event types corresponding to first nodes of all the candidate episodes. This is how the $waits(\cdot)$ list is initialized. After that, every time an event type for which some automaton is waiting occurs, that automaton makes a transition and the $waits(\cdot)$ list is accordingly updated. In addition to the $waits(\cdot)$ list, we also need to store the initialization time (i.e. time at which the first event of the episode occurred) for each instance of the automaton (for a given episode).

It useless to have multiple automata in the same state, as they would only make the same transitions. It suffices to maintain the one that reached the common state last (when counting non-overlapping occurrences). Thus, we need to store at most $|\alpha|$ initialization times for α. This is done by $\alpha.init[j]$ which indicates *when* an instance for α that is currently in its j^{th} state, got *initialized*. If multiple instances transit to the same state, we remember only the most recent initialization time.

Algorithm A, at the instant of completion of any one occurrence of the episode, resets all other automata that might have been initialized for that episode. This ensures that, any collection of overlapping occurrences increments the frequency for the episode by exactly one.

Further, it may be useful to prescribe an expiry time for episode occurrences, so that we do not count very widely spread out events as an occurrence of some episode. This condition is enforced in the algorithm by testing the time taken to reach the new state before permitting a transition into it. This expiry time condition is only an added facility in the algorithm and the condition can easily be dropped. It may be noted that the scheme of retaining only the latest initialization time for the automata in a given state is consistent with this expiry time restriction.

ALGORITHM A: NON-OVERLAPPING OCCURRENCE COUNT

Input: Set $\mathcal{C}$ of (candidate) episodes, event stream $s = \langle (E_1,t_1),\ldots,(E_n,t_n))\rangle$, frequency threshold $\lambda_{\min}$
Output: The set $\mathcal{F}$ of frequent episodes in $\mathcal{C}$

```
1: /* INITIALIZATIONS */
2: Initialize bag = φ
3: for all event types A do
4:    Initialize waits(A) = φ
5: for all α ∈ C do
6:    Add (α, 1) to waits(α[1])
7:    Initialize α.freq = 0
8:    for j = 1 to |α| do
9:       Initialize α.init[j] = 0
10: /* DATA PASS */
11: for i = 1 to |s| do
12:    for all (α, j) ∈ waits(E_i) do
13:       if j = 1 then
14:          /* INITIALIZE α AUTOMATON */
15:          Update α.init[1] = t_i
16:       else
17:          /* TRANSIT α AUTOMATON TO STATE j */
18:          Update α.init[j] = α.init[j − 1]
19:          Reset α.init[j − 1] = 0
20:          Remove (α, j) from waits(E_i)
21:       if j < |α| then
```

```
22:         if α[j+1] = E_i then
23:           Add (α, j+1) to bag
24:         else
25:           Add (α, j+1) to waits(α[j+1])
26:       if j = |α| then
27:         /* Recognize occurrence of α */
28:         Update α.freq = α.freq + 1
29:         Reset α.init[j] = 0
30:         /* Remove partial occurrences of α */
31:         for all 1 ≤ k < |α| do
32:           Reset α.init[k] = 0
33:           Remove (α, k+1) from waits(α[k+1])
34:           Remove (α, k+1) from bag
35:     Empty bag into waits(E_i)
36: /* Output */
37: for all α ∈ C such that α.freq ≥ nλ_min do
38:     Add α to the output F
```

2.2 Non-interleaved occurrences

We next present Algorithm B, which counts the number of *non-interleaved* occurrences. There are two significant changes with respect to Algorithm A. The first is that the algorithm does not permit a transition into a particular state (except the first state) if there is already an instance of the automaton waiting in that state. In other words, while it still uses only one automaton per state, it does not forget an earlier initialization of the automaton until that has transited to the next state. The second change is that we do not reset all instances of an episode if one of it reaches the final state. This way we can count some overlapping occurrences (as needed), so long as a previous instance has transited at least one state more than the next instance.

We illustrate this counting scheme with an example. Consider the same sequence that was used earlier,

$$\langle (A,1),(B,2),(A,3),(B,4),(C,5),(B,6),(C,8)\rangle.$$

Algorithm B will count *two* non-interleaved occurrences – the first with events $(A,1),(B,2),(C,5)$, and the second with events $(A,3),(B,6),(C,8)$. Note that, at time $t=4$, a second instance of the automaton for $(A \rightarrow B \rightarrow C)$ does not transit into state 2, since there is already one in that state due to the event $(B,2)$. This second instance has to wait till $t=6$, by which time the earlier instance has transited to state 3.

It may be noted that there may be many sets of *non-interleaved* occurrences of an episode in the event sequence. This algorithm counts that set of non-interleaved occurrences, which includes the first occurrence of the episode in the event sequence.

Algorithm B: Non-interleaved occurrence count

Input: Set $\mathcal{C}$ of (candidate) episodes, event stream $s = \langle (E_1,t_1),\ldots,(E_n,t_n))\rangle$, frequency threshold $\lambda_{\min}$
Output: The set $\mathcal{F}$ of frequent episodes in $\mathcal{C}$

```
 1: /* Initializations */
 2: Initialize bag = φ
 3: for all event types A do
 4:     Initialize waits(A) = φ
 5: for all α ∈ C do
 6:     Add (α, 1) to waits(α[1])
 7:     Initialize α.freq = 0
 8:     for j = 1 to |α| do
 9:         Initialize α.init[j] = 0
10: /* Data pass */
11: for i = 1 to |s| do
12:     for all (α, j) ∈ waits(E_i) do
13:         if j = 1 then
14:             /* Initialize α automaton */
15:             Update α.init[1] = t_i
16:             Set transition = 1
17:         else
18:             if (t ≠ α.init[j−1]) & (α.init[j] = 0) then
19:                 /* Transit α automaton to state j */
20:                 Update α.init[j] = α.init[j−1]
21:                 Set transition = 1
22:                 Reset α.init[j−1] = 0
23:                 Remove (α, j) from waits(E_i)
24:         if j < |α| & transition = 1 then
25:             if α[j+1] = E_i then
26:                 Add (α, j+1) to bag
27:             else
28:                 Add (α, j+1) to waits(α[j+1])
29:         if j = |α| & transition = 1 then
30:             /* Recognize occurrence of α */
31:             Update α.freq = α.freq + 1
32:             Reset α.init[j] = 0
33:     Empty bag into waits(E_i)
34: /* Output */
35: for all α ∈ C such that α.freq ≥ nλ_min do
36:     Add α to the output F
```

3. COUNTING GENERALIZED EPISODES

As was mentioned earlier in Sec. 1, the framework of frequent episode discovery has been generalized to incorporate time durations into the episode definitions [2]. In this generalized framework, the input data is a sequence of event types with (not one, but) two time stamps associated with it – a start time and an end time. Now, an episode is defined to be an ordered collection of nodes, with each node labelled not just with an event type but also with one or more time intervals. Thus, an episode in this new framework will be said to have occurred in an event sequence if the event types associated with the episode occur in the same order in the event sequence and if the duration (or dwelling time) of each corresponding event belongs to one of the intervals associated with the concerned node in the episode. For a complete treatment of the extended formalism the reader is referred to [2].

In this section, we will discuss how the new frequency counts presented in Sec. 2 extend to the case of these generalized episodes as well.

Introducing dwelling times into the episode definition does not alter the overall structure of the frequency counting algorithms. The same general counting strategy as earlier is used. Finite state automata are employed for keeping track of episodes that have occurred partially in the data while se-

quentially scanning through the event sequence. Every time we look at the next event in the event sequence, the appropriate automata are updated so that we correctly keep track of all the episodes that have occurred partially so far. Also, whenever the next event results in the full occurrence of an episode we increment its frequency count appropriately.

In order to describe episodes with event durations, for each episode α, we now need an $\alpha.g[i]$ which denotes the event type associated with its i^{th} node and an $\alpha.d[i]$ which denotes the set of time intervals associated with its i^{th} node. Even if an automaton, of say α, is waiting in its i^{th} node for event type A (i.e. $\alpha.g[i] = A$) then a next A in the event sequence does not necessarily transit this automaton to its next state. This will happen only if this event A in the sequence has a dwelling time that satisfies the constraints imposed by the contents of $\alpha.d[i]$. Hence, the *waits* list, which links together all automata that accept a particular event, will now need to be indexed by an ordered pair (A, δ) rather than by just the event type A, as was the case earlier. The set $waits(A, \delta)$ stores (α, i) pairs, indicating that an automaton for episode α is waiting in its i^{th} state for the event type A to occur with a duration in the time interval, δ, so that it can make a transition to the next node of the episode.

Another difference with respect to the earlier counting algorithms is that since we now have a start and end time for each event in the event sequence, either one of them may be stored in $\alpha.init[j]$ when an automaton for α transits to the j^{th} state. Basically, it is a matter of convention whether an event is regarded as having occurred at its start time or its end time.

Algorithm A1 below shows how to count non-overlapped occurrances of generalized episodes. The algorithm for non-interleaved occurrances is much along the same lines and hence has been omitted for brevity.

ALGORITHM A1: NON-OVERLAPPING OCCURRENCE COUNT FOR GENERALIZED EPISODES

Input: Set $\mathcal{C}$ of (candidate) episodes, event stream $s = \langle (E_1, t_1, \tau_1), \ldots, (E_n, t_n, \tau_n)) \rangle$, frequency threshold $\lambda_{\min}$, set $\mathbf{B}$ of allowed time intervals
Output: The set $\mathcal{F}$ of frequent episodes in $\mathcal{C}$

/* INITIALIZATIONS */
Initialize $bag = \phi$
for all event types A and time intervals δ **do**
  Initialize $waits(A, \delta) = \phi$
for all $\alpha \in \mathcal{C}$ **do**
  Add $(\alpha, 1)$ to $waits(\alpha.g[1], \delta)$ $\forall \delta \in \alpha.d[1]$
  Initialize $\alpha.freq = 0$
  for $j = 1$ to $|\alpha|$ **do**
    Initialize $\alpha.init[j] = 0$
/* DATA PASS */
for $i = 1$ to $|s|$ **do**
  Choose $D \in \mathbf{B}$ s.t. $D.\text{left} \leq (\tau_i - t_i) \leq D.\text{right}$
  for all $(\alpha, j) \in waits(E_i, D)$ **do**
    if $j = 1$ **then**
      /* INITIALIZE α AUTOMATON */
      Update $\alpha.init[1] = t_i$
    else
      /* TRANSIT α AUTOMATON TO STATE j */
      Update $\alpha.init[j] = \alpha.init[j-1]$
      Reset $\alpha.init[j-1] = 0$
      Remove (α, j) from $waits(E_i, \delta)$ $\forall \delta \in \alpha.d[j]$
    if $j < |\alpha|$ **then**
      if $\alpha.g[j+1] = E_i$ and $D \in \alpha.d[j+1]$ **then**
        Add $(\alpha, j+1)$ to *bag*
        Add $(\alpha, j+1)$ to $waits(\alpha[j+1], \delta)$ $\forall \delta \in \alpha.d[j+1], \delta \neq D$
      else
        Add $(\alpha, j+1)$ to $waits(\alpha[j+1], \delta)$ $\forall \delta \in \alpha.d[j+1]$
    if $j = |\alpha|$ **then**
      /* RECOGNIZE OCCURRENCE OF α */
      Update $\alpha.freq = \alpha.freq + 1$
      Reset $\alpha.init[j] = 0$
      /* REMOVE PARTIAL OCCURRENCES OF α */
      for all $1 \leq k < |\alpha|$ **do**
        Reset $\alpha.init[k] = 0$
        Remove $(\alpha, k+1)$ from $waits(\alpha.g[k+1], \delta)$ $\forall \delta \in \alpha.d[k+1]$
        Remove $(\alpha, k+1)$ from *bag*
  Empty *bag* into $waits(E_i, D)$
/* OUTPUT */
for all $\alpha \in \mathcal{C}$ such that $\alpha.freq \geq n\lambda_{\min}$ **do**
  Add α to the output $\mathcal{F}$

---

4. RESULTS

This section presents some results obtained with the new frequency counting algorithms. We quote results obtained on both synthetically generated data as well as manufacturing plant data from General Motors.

The synthetic data is generated as some random time series containing temporal patterns embedded in noise. The temporal patterns that were picked up by our algorithms correlated very well with those that were used to generate the data even when these patterns are embedded in varying amounts of noise. Also, the set of frequent episodes discovered is essentially the same as that discovered using the algorithms form [1]. We also show that our algorithms results in a speed-up by a factor of 7 or more.

4.1 Synthetic data generation

Each of the temporal patterns to be embedded consists of a specific ordered sequence of events. A few such temporal patterns are specified as input to the data generation process which proceeds is as follows. There is a counter that specifies the current time instant. Each time an event is generated, it is time-stamped with this current time as its start time. Further, another random integer is generated which is then associated as this event's dwelling time. The end time of the event is now computed by adding the start and dwelling times and the current time is set to equal this end time. After generating an event (in the event sequence) the current time counter is incremented by a small random integer. Each time the next event is to be generated, we first decide whether the next event is to be generated randomly with a uniform distribution over all event types (which would be called an *iid* event) or according to one of the temporal

patterns to be embedded. This is controlled by the parameter ρ which is the probability that the next event is *iid*. If $\rho = 1$ then the data is simply *iid* noise with no temporal patterns embedded. If it is decided that the next event is to be from one of the temporal patterns to be embedded, then we have a choice of continuing with a pattern that is already embedded partially or starting a new occurrence of one of the patterns. This choice is also made randomly. It may be noted here that due to the nature of our data generation process, embedding a temporal pattern is equivalent to embedding many episodes. For example, suppose we have embedded a pattern $A \rightarrow B \rightarrow C \rightarrow D$. Then if this episode is frequent in our event sequence then, based on the amount of noise and our expiry time constraints (in the counting algorithm), episodes such as $B \rightarrow C \rightarrow D \rightarrow A$ can also become frequent.

4.2 Results on synthetic data

In this section we present a few of our simulation results to compare our algorithms (i. e. Algorithm A and Algorithm B) with the windows-based frequency count of [1] (which is referred to as Algorithm C in the tables below).

We first demonstrate the effectiveness of our algorithms by comparing the frequent episodes discovered when the event sequence is *iid* with those when we embed some patterns. Suppose we take $\rho = 1$. Then event types and dwelling times are chosen randomly from a uniform distribution. Hence we expect any sequence of, e.g., two events to be as frequent in the data as any other sequence of two events. Thus, if we are considering all 2-node episodes then most of them would have similar frequencies. If we increase the frequency threshold starting from a low value, initially most of the episodes would be frequent and, after some critical threshold, most of them would not be frequent. Now suppose we embed a few temporal patterns. Then some of its permutations and all their subepisodes would have much higher frequencies than other episodes. Hence if we plot the number of frequent (principle) episodes found versus frequency threshold, then, in the *iid* case we should see a sudden drop in the graph while in the case of data with embedded patterns, the graph should level off. Fig. 1 shows the plot of the number of 2-node frequent episodes discovered by the algorithm versus the frequency threshold in the two cases of *iid* data and biased data with the number of non-overlapping occurrences being the frequency count. Fig. 2 shows the same things for non-interleaved occurrences frequency count. For the biased data we put in two 4-node patterns so that sufficient number of 2-node episodes would be frequent. In both the graphs, the sudden transition in case of *iid* event sequences is very evident. Similar results were obtained for longer episodes as well. It is noted here that in the iid case, since all permutations of events are equally likely, the number of frequent episodes (that meet a low frequency threshold criterion) is much higher than in biased data.

In the next experiment we describe, data was generated by embedding two patterns in varying degrees of *iid* noise. The two patterns that we embedded, are as follows: (1) $\alpha = (B \rightarrow C \rightarrow D \rightarrow E)$ and (2) $\beta = (I \rightarrow J \rightarrow A)$. Data sequences with 5000 events each were generated for different values of ρ, namely $\rho = 0.0, 0.2, 0.3, 0.4$ & 0.5. The objective is to see whether these two patterns indeed appeared among

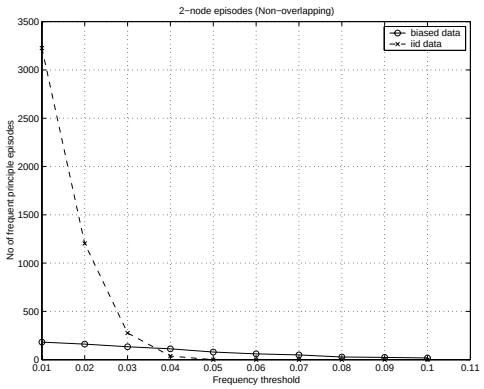


Figure 1: Effect of frequency threshold (Non-overlapping occurrences) using simulated data: 2-node episodes

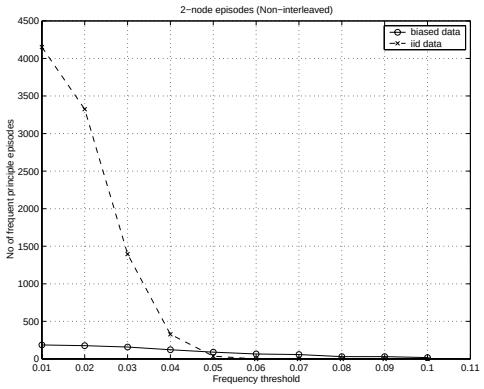


Figure 2: Effect of frequency threshold (Non-interleaved occurrences) using simulated data: 2-node episodes

the set of frequent episodes discovered, and if so, at what position. Since α is a 4-node pattern and β is a 3-node pattern, their respective positions (referred to as their *ranks*) in the (frequency) sorted 3-node and 4-node frequent episode sets discovered are shown in Tables 1–2. As can be seen from the tables, our frequency counts are as effective as the windows-based frequency proposed in [1].

We next compare the time complexity of the different algorithms by looking at the overall run times for frequent episode discovery. We have taken data sequences of length 5000 and chosen the frequency thresholds so as to get roughly the same number (around 50) of 4-node frequent episodes. We have varied ρ from 0.0 to 0.5. These results are shown in Table 3. The last column of the table gives the speed up achieved by Algorithm A in comparison with Algorithm C.

It is seen from the table that the algorithms based on counting the number of non-overlapping occurrences and the number of non-interleaved occurrences, run much faster than the windows-based frequency counting algorithm. This advantage comes from the fact that we are counting occurrences and not windows. In the windows-based count, in addition

ρ	Algo A	Algo B	Algo C
0.0	1	1	1
0.2	1	1	1
0.3	1	1	1
0.4	1	11	1
0.5	1	19	1

Table 1: Rank of α in sorted 4-node frequent episodes set

ρ	Algo A	Algo B	Algo C
0.0	1	1	5
0.2	1	1	5
0.3	6	29	5
0.4	5	27	7
0.5	3	4	6

Table 2: Rank of β in sorted 3-node frequent episodes set

to keeping track of new events that enter each sliding window, one also has to keep track of any events falling out of it at its left extremity. This has an additional temporary memory overhead as well since we need a list that stores for each time the automata which make transitions at that particular time. Moreover, this necessitates checking for new events and events that fall out at every time tick. In contrast, our occurrence-based counts need to act only every time a new event occurs in the sequence. This property translates to major run-time gains if the number of events is much less than the actual time span of the data sequence.

Also, the noise in the data, per se, has no effect on the runtime as can be expected from the algorithms. Through another set of simulations, it is observed that the run times for all three frequency counts increase roughly at the same rate with the number of events in the data sequence. This is shown in Table 4 where run times for data of length 5000 and 20000 events are compared. All the run times seem to scale roughly linearly with the data length (which is what we expect from the algorithms).

4.3 Results on GM data

This section describes some experiments on GM data. This data pertains to stamping plants that make various body parts of cars. Each plant has one or more stamping lines. The data is the time-stamped logs of the status of these lines. Each event is described by some breakdown codes

ρ	Algo A	Algo B	Algo C	Speed-up
0.0	22	22	156	7.1
0.2	22	21	271	12.3
0.3	21	21	144	6.9
0.4	21	21	236	11.2
0.5	21	21	193	9.2

Table 3: Run-times (in seconds) for different noise levels

Length	Algo A	Algo B	Algo C	Speed-up
5000	22	22	156	7.1
20000	87	117	625	7.2

Table 4: Run-times (in seconds) for different sequence lengths

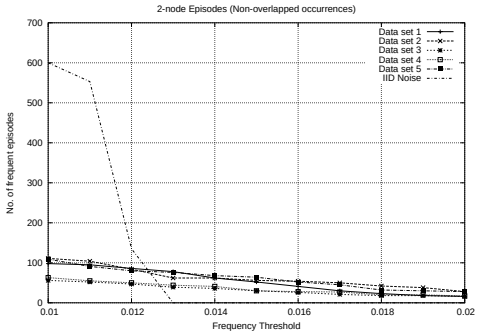


Figure 3: Effect of frequency threshold (Non-overlapping occurrences) using GM data: 2-node episodes

when the line is stopped due to some problem or by a code to indicate the running status. The objective of analysis is to find frequent episodes that can throw light on frequent co-occurring faults.

In Section 4.2 we demonstrated how the graphs for the number of episodes discovered versus frequency threshold falls in characteristically different ways for iid data and data with some patterns embedded in them. We can use these to ask whether the GM data has any patterns of interest at all. In order to do this we plotted the number of frequent episodes discovered (as a function of frequency threshold) on the GM data. The number of different breakdown codes in these sequences were roughly around 25. So, for comparison we also plot the number of frequent episodes obtained on an iid sequence with 25 event types of length 50000 (The lengths of data slices from GM data we analyzed were also of the same order.). As before, plots were obtained for both the frequency counts. Fig. 3 gives the graph for the frequency count based on non-overlapped occurrences and Fig. 4 gives the graphs based on non-interleaved occurrences. These plots closely resemble the earlier plots on simulated data i.e. Fig. 1 and Fig. 2. Again this experiment was repeated for episodes of larger sizes too. We may infer from these that the GM data on which the algorithms were run indeed contained patterns with some strong temporal correlations quite unlike the case of iid data.

Our new frequency counts made fast exploration of the large data sets from GM plants feasible and some interesting temporal patterns were obtained.

4.4 Conclusions

From the results described, we can conclude that in all cases, our frequency counting algorithms result in a speed-up by a factor of 7 while delivering similar quality of output. It

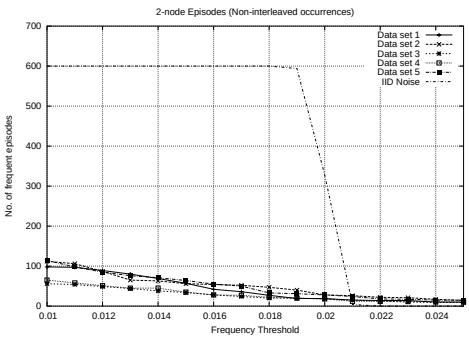


Figure 4: Effect of frequency threshold (Non-interleaved occurrences) using GM data: 2-node episodes

may be recalled that all three frequency counts ensure that there is at most one automaton per state per episode during the frequency counting process. Thus, the space complexity of the three algorithms is of the same order. However, in terms of actual amount of temporary storage needed, the windows-based frequency count is a little more expensive because it has to keep the list *beginsat* which is not needed in the other two algorithms (See [1] for the details of why this list is needed in the windows-based frequency count).

5. DISCUSSION

Mining of interesting temporal patterns from data which is in the form of time series of events is an important data-mining problem. The framework of frequent episodes introduced in [1] is very useful for this purposes. In this paper we proposed two attractive alternatives to the somewhat non-intuitive windows-based frequency measure of [1].

Unlike in the static data-mining scenario, whether or not a temporal pattern occurs cannot be ascertained by looking at only one record at a time in a memoryless fashion. Hence for recognizing the occurrence of episodes we need finite state automata. In this context we have explained why it would be very inefficient if we want to count *all* occurrences of episodes. Based on the insight gained, we suggested two possible ways to restrict counting to only some specialized occurrences. Both of these frequency counts can be achieved with a reasonable and fixed number of automata per episode so that the space complexity is controlled. In particular, the number of automata needed here are the same as those needed in the windows-based count proposed in [1]. It is also shown through simulations that both these frequency measures are much more efficient in terms of the time taken and they are just as effective in discovering frequent episodes as the windows-based frequency counting algorithm. Thus, these new algorithms make the framework of frequent episode discovery attractive in many more applications.

6. REFERENCES

[1] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of frequent episodes in event sequences," *Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 259–289, 1997.

[2] S. Laxman, P. S. Sastry, and K. P. Unnikrishnan, "Generalized frequent episodes in event sequences," in *Temporal Data Mining Workshop Notes* (K. Unnikrishnan and R. Uthurusamy, eds.), (Edmonton, Alberta, Canada), 2002.

[3] M. Last, Y. Klein, and A. Kandel, "Knowledge discovery in time series databases," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 31, pp. 160–169, Feb. 2001.

[4] M. L. Hetland and P. Strom, "Temporal rule discovery using genetic programming and specialized hardware," in *Proc. of the 4th Int. Conf. on Recent Advances in Soft Computing (RASC)*, 2002.

[5] R. Agrawal and R. Srikant, "Mining sequential patterns," in *11th Int'l Conference on Data Engineering, Taipei, Taiwan*, Mar. 1995.

[6] Z. Tronicek, "Episode matching," in *Combinatorial Pattern Matching*, pp. 143–146, 2001.

[7] M. Hirao, S. Inenaga, A. Shinohara, M. Takeda, and S. Arikawa, "A practical algorithm to find the best episode patterns," *Lecture Notes in Computer Science*, vol. 2226, pp. 435–441, 2001.

[8] G. Casas-Garriga, "Discovering unbounded episodes in sequential data," in *7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03). Cavtat-Dubvrovnik, Croatia.*, 2003.

[9] S. Laxman, P. S. Sastry, and K. P. Unnikrishnan, "Fast algorithms for frequent episode discovery in event sequences," Tech. Rep. CL-2004-04/MSR, GM R&D Center, Warren, 2004.

Extracting time-ordered pairs of similar subsequences by time warping approach

Souhei Morita
Keio Research Institute at SFC
5322 Endo Fujisawa
Kanagawa, Japan

Koichi Furukawa
Graduate School of Media and Governance, Keio University
5322 Endo Fujisawa
Kanagawa, Japan

Tomonobu Ozaki
Graduate School of Media and Governance, Keio University
5322 Endo Fujisawa
Kanagawa, Japan

E-mail: {souhei, furukawa, tozaki}@sfc.keio.ac.jp

Abstract

We present a novel algorithm, called "Paths Stitching (PS)", to extract time-ordered pairs of similar subsequences of variable length from a pair of time series data. Since the PS algorithm uses Dynamic Time Warping (DTW) as a matching procedure, this algorithm is robust for a locally fluctuant time-difference between subsequences. We also introduce a new lower bounding function suitable for the PS algorithm. Lower bounding functions are recently used to avoid futile DTW execution as far as possible.

Although similarity search is a very popular approach for time series analysis, techniques to extract pairs of similar subsequences of variable length are not researched much. However, we think there are many unfamiliar domains in which they are useful as preprocessing. One domain in which our algorithm is very effective is in eye movement analysis. By applying the PS algorithm to an eye movement analysis, we demonstrate its feasibility.

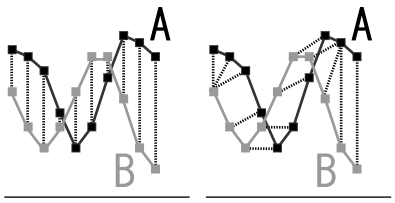
1. Introduction

Similarity search is one of the most popular approaches for time series analysis. Similarity between two time sequences generally depends on the distance between them. If this distance is short, we regard two sequences as similar. By using similarity search, we are able to retrieve (sub)sequences that are similar to a given query or find pairs of similar sequences in a database. Plenty of similarity search applications have been developed to attain each or both of these purposes[1, 5, 6, 8].

However, in this paper, we are interested in a less familiar application of similarity search. We present a novel algorithm which extracts time-ordered pairs of similar subsequences of variable length from a pair of time sequences. We call our proposed algorithm "Paths Stitching (PS)". Agrawal et al. have already proposed a technique available for the same purpose[2]. In distance calculation phases, Agrawal's popular technique allows amplitude scaling, offsetting translation, offsetting global time-differences and removing outliers included in subsequences. On the other hand, in the same phase, the PS algorithm allows nonlinear time-normalization by employing Dynamic Time Warping (DTW) as a matching procedure. DTW is a pattern matching algorithm absorbing local time-differences between two time sequences. Because of this specialty, DTW seems to be a very useful algorithm. But DTW is expensive in terms of computational time. To overcome this defect, recently several techniques adopting a lower bounding function to reduce futile DTW execution are introduced[7, 10, 12]. In this paper, we introduce a new lower bounding function suitable for our algorithm.

Subsequences exist at any position within a sequence. However, it is obviously undesirable to carry out matching computation for all possible pairs of subsequences and acquire all pairs of subsequences that are similar. Such process would spend an enormous amount of time and cause a very redundant output. Therefore, we need conditions qualifying the output. We introduce two simple conditions: the output must be time consistent and must not be redundant. These two conditions are intuitive and powerful.

Although techniques to extract pairs of similar subsequences of variable length have not been paid much attention to, they have several advantages over major applications of similarity search. These techniques enable us to do



(a) (b)

Figure 1. An illustration of distances between elements. They are used for calculation of: (a) the Euclidean distance. (b) the DTW distance.

more delicate analysis than the techniques which extract pairs of similar sequences. Moreover, we can get truly unexpected pairs of similar subsequences, unlike the case of using a query.

The organization of the rest of the paper is as follows. In Section 2 we review the DTW algorithm, which is a main component of the PS algorithm. The PS algorithm employs DTW as a matching procedure. In Section 3 we present the PS algorithm which extracts time-ordered pairs of similar subsequences of variable length. Then, we introduce a new lower bounding function suitable for the PS algorithm in section 4. In Section 5 we show experimental results to demonstrate the feasibility of our algorithm. Finally, we summarize and conclude this work in Section 6.

2. Dynamic Time Warping

Most distance functions employed to measure the similarity utilize distances between an element of one sequence and one of the other. The DTW distance function is also not an exception. The specialty of the DTW approach is that it can offset a locally fluctuant time-difference between sequences (figure 1). This specialty causes a more intuitively correct distance than the Euclidean distance function which is the most popular function for similarity measure.

DTW is a pattern matching algorithm. The basic idea of this algorithm was proposed for spoken word recognition[11]. Recently, the data mining community has also paid attention to DTW[3]. Next we review the DTW algorithm.

2.1 The DTW Algorithm

Suppose two sequences A and B, each length is I and J.

$$A = a_1, a_2, \ldots, a_i, \ldots, a_I \quad (1)$$

$$B = b_1, b_2, \ldots, b_j, \ldots, b_J \quad (2)$$

In order to calculate the DTW distance between A and B, let us consider the $I \times J$ plane. A cell (i, j) on the plane contains a distance $d(a_i, b_j)$ between a_i and b_j. Distances between elements are generally calculated as the Euclidian distance. Under a specific distance measurement between

A	A sequence of length I.
B	A sequence of length J.
$d(a_i, b_j)$	A distance between element a_i and b_j.
$D_{tw}(A, B)$	A DTW distance between sequence A and B .
W	A path on the $I \times J$ plane.
w_k	k^{th} element of W.
ω	A warping window width
s	The maximum length of horizontal/vertical segments
$\gamma(i, j)$	A cumulative distance from (1,1) to (i,j)
l	The lower bound of subsequence length
ϵ	The threshold for similarity-based matching
W^l	A local path.
W^g	A global path.
W^c	A candidate of a global path.
$LB(A, B)$	A lower bounding function of a sequence pair(A, B).

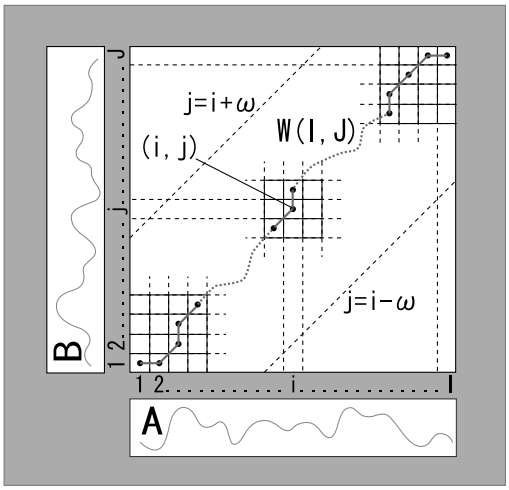
Table 1. A symbol list.

elements and several conditions, DTW calculates a distance $D_{tw}(A, B)$ between two sequences A and B. DTW can be regarded as a problem of making an optimum warping path $W(I, J)$ on the $I \times J$ plane (figure 2).

$$W(I, J) = w_1, w_2, \ldots, w_k, \ldots, w_K \quad (3)$$
$$w_k = (i_k, j_k)$$

$W(I, J)$ is a sequence. Each of W's elements is a cell on the $I \times J$ plane. $D_{tw-average}(A, B) = \frac{\sum_{k=1}^{K} d(a_{i_k}, b_{j_k})}{K}$ or $D_{tw-max}(A, B) = \max_{k=1}^{K} d(a_{i_k}, b_{j_k})$ is typically employed as $D_{tw}(A, B)$. Generally, the shape of $W(I, J)$ is restricted by the following conditions. We expediently define $i_0 = j_0 = 0$. We suppose $1 \leq k \leq K$.

- Monotonic conditions: $i_{k-1} \leq i_k$ and $j_{k-1} \leq j_k$ must be held.
- Continuity conditions: w_k is allowed to connect only adjacent elements. Thus, $i_k - i_{k-1} \leq 1$, $j_k - j_{k-1} \leq 1$.
- Global constraint condition: An element of W is bounded to a specific region on the $I \times J$ plane. Sakoe and Chiba[11] who initially made public defined the region by inequality $|i - j| \leq \omega$. ω is called "warping window width". PS algorithm also employs warping window width as a global constraint condition.
- Slope constraint condition: Too steep or too gentle a gradient should not be allowed for warping path. s denotes maximum length of horizontal/vertical segments of paths restricted by a slope constraint condition.



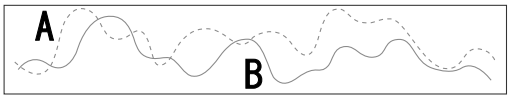


Figure 2. An example of a warping path $W(I, J)$.

- Boundary conditions: These conditions fix a start point and end point of warping paths. $w_1 = (1, 1)$, $w_K = (I, J)$.

The PS algorithm adopts all the above conditions when it uses DTW.

Although there are many paths holding the above conditions, we can find the best path which has a minimum $D_{tw}(A, B)$ distance by using a dynamic programming technique. Let $\gamma(i, j)$ be a cumulative distance of (i, j). $\gamma(i, j)$ is defined as:

$$\gamma(i,j) = \min \begin{pmatrix} F(\gamma(i-1,j), d(a_i,b_j)) \\ F(\gamma(i,j-1), d(a_i,b_j)) \\ F(\gamma(i-1,j-1), d(a_i,b_j)) \end{pmatrix} \quad (4)$$

Where $\gamma(i-1, j)$, $\gamma(i, j-1)$, $\gamma(i-1, j-1)$ are cumulative distances of adjacent cells of (i, j). We expediently define that $\gamma(0, 0) = 0, \gamma(1, 0) = \gamma(0, 1) = \infty$. By solving equation 4 recurrently, we can calculate $\gamma(I, J)$. $D_{tw}(A, B) = \gamma(I, J)$.

Slope constraint conditions are implemented by modifying equation 4. So if we use slope constraint conditions, we must change elements referred in equation 4 as a function of s. Sakoe and Chiba adopted "*simplified path*" when they implemented a slope constraint condition[11]. Simplified paths are banned from orthogonal changing its direction and they implicitly have a shape restriction that the first segment of paths must be a diagonal line. The PS algorithm also adopts simplified paths when it uses DTW.

As an example of a simplified path, we illustrate the equation and the diagram of $s = 1$ (table 2, figure 3-b). Figure 3-a shows the diagram of no slope constraint condition, which is the case of equation 4.

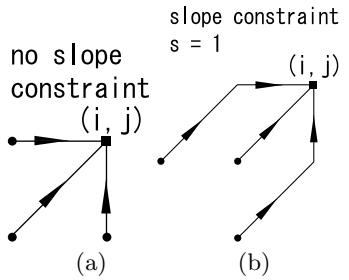


Figure 3. A diagram of slope constraint conditions.

$$\gamma(i,j) = \min \begin{pmatrix} F(\gamma(i-2,j-1), d(a_{i-1},b_j), d(a_i,b_j)) \\ F(\gamma(i-1,j-2), d(a_i,b_{j-1}), d(a_i,b_j)) \\ F(\gamma(i-1,j-1), d(a_i,b_j)) \end{pmatrix}$$

Table 2. The equation of the case of $s = 1$.

Time complexity of DTW is $O(IJ)$. If we employ the global constraint that Sakoe and Chiba proposed, time complexity becomes $O(I\omega)$ or $O(J\omega)$. Because DTW allows nonlinear time-normalization, it is possible to calculate similarity of sequences of different lengths.

3. The Paths Stitching Algorithm

In this section, we present a novel algorithm to extract time-ordered pairs of similar subsequences of variable length from a pair of sequences. We want to output the best sequences of pairs of similar subsequences under our own evaluation measure. By comparing all sequences of pairs of similar subsequences, we can get a desirable output. But this brute algorithm is clearly impractical because it spends enormous time. Our algorithm proposes to get a desirable output more quickly. We call it the Paths Stitching (PS) algorithm.

3.1 Formulation

We consider an extraction from a sequence pair (A, B). We define paths that keep to the following four conditions as "local path". We represent a local path as:

$$W^l = (i_1, j_1),...,(i_m, j_m),...,(i_M, j_M) \quad (5)$$

1. W^l must be produced by DTW of input $A[i_1 : i_M], B[j_1 : j_M]$. Where $A[x : y] = (a_x,...,a_y)$.
2. A subsequences pair which corresponds to a local path must be similar. Thus $D_{tw}(A[i_1 : i_M], B[j_1 : j_M]) \leq \epsilon$. We employ the maximum distance $D_{tw-max}(A', B')$ as the DTW distance $D_{tw}(A', B')$.
3. We define the lower bound of length of subsequences that constitute a local path as $l(> 1)$. Thus $(i_M - i_1 + 1) \geq l \wedge (j_M - j_1 + 1) \geq l$.
4. We use simplified paths when we make W^l by DTW. Thus, any first segment of local paths must be a diagonal line. That is $i_2 - i_1 = j_2 - j_1 = 1$.

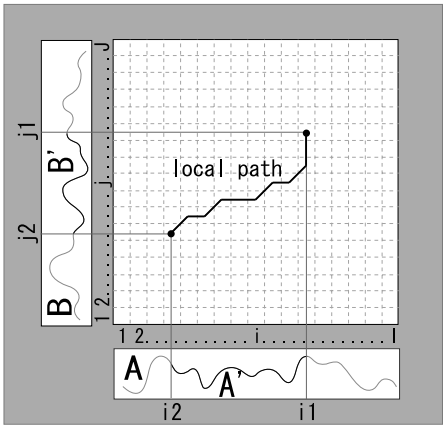


Figure 4. Relationship between a pair of similar subsequences (A', B') and a local path.

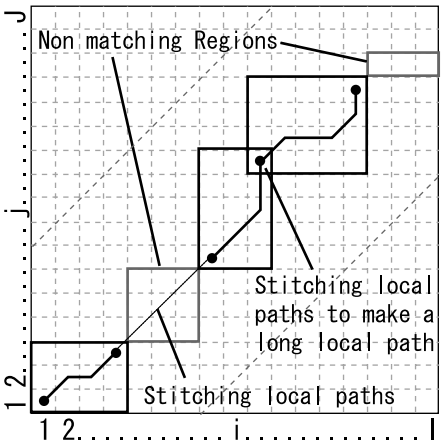


Figure 5. An example of stitching to make a global path.

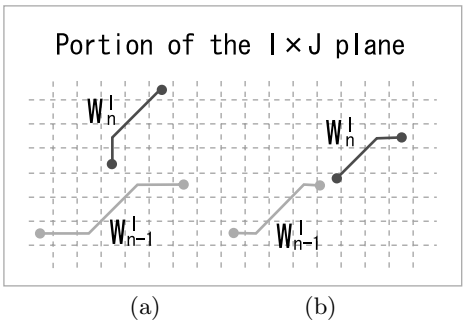


Figure 6. Simple examples of: (a) inconsistency. (b) redundancy.

Any pair of subsequences, of lengh l or longer than l, included in (A, B) is represented as a local path on the $I \times J$ plane if they are similar (figure 4).

The concept of the PS algorithm is to stitch local paths one after another on the $I \times J$ plane as in figure 5. There is often a non-matching region between stitched local paths. To represent this behavior, we call our proposed algorithm "Paths Stitching". We call a sequence of stitched local paths "global path" and represent it as W^g. We also represent n^{th} element of W^g (i.e. a local path) as W_n^l.

$$W^g = W_1^l,...,W_n^l,...,W_N^l \tag{6}$$

$$W_n^l = (i_1^n, j_1^n),...,(i_m^n, j_m^n),...,(i_M^n, j_M^n) \tag{7}$$

The output of the PS algorithm is one global path.

3.2 Algorithm Detail

There would be many local paths in the $I \times J$ plane. Thus there would also be many global paths. To speed up calculation and to achieve a meaningful result, we impose two constraint conditions for global paths (figure 6). First, global paths must be time consistent. We define that global paths which hold equation 8 are time consistent.

$$\forall n \ \ (i_M^{n-1} \leq i_1^n) \wedge (j_M^{n-1} \leq j_1^n) \tag{8}$$

Second, global paths must not be redundant. We also define that global paths which hold equation 9 are not redundant.

$$\neg \exists n \ \ s.t. \ \ (i_M^{n-1} = i_1^n) \wedge (j_M^{n-1} = j_1^n) \tag{9}$$

For these two conditions, we expediently define:

$$(i_M^0, j_M^0) = (0, 0) \tag{10}$$

In order to keep to "no redundancy" condition, the PS algorithm makes a long local path if two stitched local paths overlap on the edges (see figure 5).

A cell (i, j) on the $I \times J$ plane stores the best global path in the $i \times j$ plane. We represent this global path as $W^g(i, j)$. To decide the global path which (i, j) stores and to decide the output, the PS algorithm has to evaluate global paths in several phases. We define the value of a global path as

the following formula. We regard that the smaller the value the more superior the global path $W^g(i,j)$.

$$\sum_{n=1}^{N}\sum_{m=1}^{M} d(a_{i_m^n}, b_{j_m^n}) + (\sum_{n=1}^{N}(i_1^n - i_M^{n-1} + j_1^n - j_M^{n-1} - 2) + (i - i_M^N + j - j_M^N)) * \delta \quad (11)$$

Where δ is a given number greater than the threshold ϵ for judging similarity. The first member of equation 11 stands for a summation of distances between elements of local paths which are included in $W^g(i,j)$. "$(i_1^n - i_M^{n-1} + j_1^n - j_M^{n-1} - 2)$" denotes a summation of height and width of a non-matching region. "$i - i_M^N + j - j_M^N$" also denotes that of the non-matching region between the last local path and (i,j). We suppose equation 10 holds when we calculate equation 11. Equation 11 enables us to select the global path that has larger matching regions.

The PS algorithm outputs a global path of a minimum value among $W^g(i',J)$ and $W^g(I,j')$, where i' and j' hold the following formulas:

$$\max(1, J-\omega) \le i' \le \min(I, J+\omega) \quad (12)$$

$$\max(1, I-\omega) \le j' \le \min(J, I+\omega) \quad (13)$$

ω is the warping window width. The PS algorithm recurrently calculates all necessary global paths from $W^g(1,1)$ to $W^g(\min(I, J+\omega), \min(J, I+\omega))$. Next, we describe about the procedure of global path calculation.

Global Path Calculation

The PS algorithm stitches local paths to obtain a global path $W^g(i,j)$. If local paths overlap each other, we make a long local path by stitching them. Below we give a complete picture of how to obtain $W^g(i,j)$ by stitching local paths.

First, let us consider cells (i_{start}, j_{start}) and (i_{end}, j_{end}) on the $I \times J$ plane. They hold the following conditions:

- $i_{end} > i_{start}$ and $j_{end} > j_{start}$.
- There is a local path starting from (i_{start}, j_{start}) to (i_{end}, j_{end}). $W^l_{start\sim end}$ denotes this local path.

Next, we consider the condition under which there is a possibility that a local path $W^l_{start\sim end}$ *can not be acquired by stitching two local paths* $W^l_{start\sim stitching}$ *and* $W^l_{stitching\sim end}$. The two local paths overlap on the stitching point, that is the edges, $(i_{stitching}, j_{stitching})$. Because the distance between subsequences (i.e. D_{tw-max}) is a monotone increasing function and the distance which corresponds to $W^l_{start\sim end}$ is not greater than the threshold, both distances which correspond to $W^l_{start\sim stitching}$ or $W^l_{stitching\sim end}$ can not be greater than the threshold. Thus, it is obvious that if we have two local paths they holds:

$$l_{ss} \equiv \min \begin{pmatrix} (i_{end} - i_{stitching} + 1) \\ (j_{end} - j_{stitching} + 1) \\ (i_{stitching} - i_{start} + 1) \\ (j_{stitching} - j_{start} + 1) \end{pmatrix}$$

$$l \le l_{ss} \quad (14)$$

we can acquire $W^l_{start\sim end}$ by stitching. Where l_{ss} denotes the length of the shortest subsequence. What situation is this condition built upon? To think about this situation, we start with a definition that $l_{shorter}$ denotes the length of the shorter subsequence constituting $W^l_{start\sim end}$. l_{longer} denotes the length of the longer subsequence.

$$l_{shorter} = \min \begin{pmatrix} (i_{end} - i_{start} + 1) \\ (j_{end} - j_{start} + 1) \end{pmatrix} \quad (15)$$

$$l_{longer} = \max \begin{pmatrix} (i_{end} - i_{start} + 1) \\ (j_{end} - j_{start} + 1) \end{pmatrix} \quad (16)$$

It simply seems that if $l_{shorter}$ is only greater than $2l-1$, we have the two local paths which hold equation 14. But this is incorrect because we have the shape restriction of local paths that the first segments should be diagonal. By the shape restriction, to guarantee the existence of two local paths which can make $W^l_{start\sim end}$ by stitching, l_{ss} must be equal to or more than $l+s$ (see figure 7). Therefore the conditions which guarantee the acquisition by stitching are:

$$l_{shorter} \ge 2(l+s) - 1 \quad (17)$$

Thus acquisition by stitching is not guaranteed if we have the following formula:

$$l_{shorter} \le 2(l+s) - 2 \quad (18)$$

Next we consider the upper bound of l_{longer}. We must give consideration to the shape restriction, once again. If it were not for the shape restriction, we have the formula: $l_{longer} \le l_{shorter} \times (s+1)$. In fact because every first segment of local paths must be diagonal, we have the following formula (see figure 8):

$$l_{longer} \le l_{shorter} \times (s+1) - s \quad (19)$$

equation 18 and equation 19 suggest that if we have equation 20, there is a possibility that we can not acquire $W^l_{start\sim end}$ by stitching.

$$l_{longer} \le \{2(l+s)-2\} \times (s+1) - s \quad (20)$$

By contrary, if we have:

$$l_{longer} > \{2(l+s)-2\} \times (s+1) - s \quad (21)$$

We can always acquire it by stitching. We have Lemma 1 regarding to stitch local paths.

Lemma 1: If (i_{start}, j_{start}) and (i_{end}, j_{end}) satisfy equation 22, it is guaranteed that we can acquire $W_{start\sim end}$ by stitching two local paths.

$$\max \begin{pmatrix} i_{end} - i_{start} + 1 \\ j_{end} - j_{start} + 1 \end{pmatrix} > \{2(l+s)-2\} \times (s+1) - s \quad (22)$$

We derive Theorem 1 from Lemma 1.

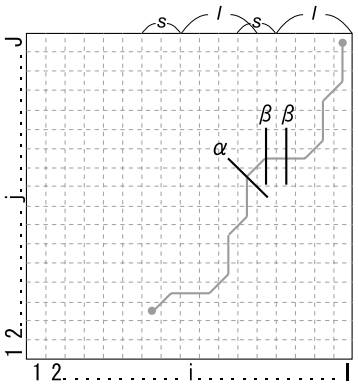


Figure 7. An illustration of the shape restriction. It is possible to acquire the local path by stitching at α but impossible at β because there is the shape restriction of local paths.

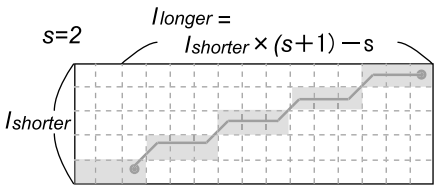


Figure 8. An illustration of the upper bound of l_{longer}.

Theorem 1: Let us consider a square region, $\{2(l+s)-2\} \times (s+1) - s$ on a side. The top right corner cell of the square region is (i_1, j_1). We represent any cell except (i_1, j_1) in this region as (i_2, j_2). If $i_2 \leq 0$ or $j_2 \leq 0$, we don't take into consideration its cell for the rest of this procedure.

When we have all $W^g(i_2, j_2)$, we can calculate $W^c(i_1, j_1, i_2, j_2)$ every (i_2, j_2) by the following procedure. The way to calculate $W^c(i_1, j_1, i_2, j_2)$ varies by whether or not there is a local path W^l_{temp} starting from (i_2, j_2) to (i_1, j_1).

If there is W^l_{temp}: We stitch W^l_N, the last element of $W^g(i_2, j_2)$, and W^l_{temp}. If (i^N_M, j^N_M), the last element of W^l_N, equals (i_2, j_2), a long local path is produced by this stitching. We replace W^l_N in $W^g(i_2, j_2)$ with this new long local path, and regard this new global path as $W^c(i_1, j_1, i_2, j_2)$. If $(i^N_M, j^N_M) \neq (i_2, j_2)$, this stitching results in adding W^l_{temp} to $W^g(i_2, j_2)$. We regard the new global path as $W^c(i_1, j_1, i_2, j_2)$.

If there is notW^l_{temp}: We regard $W^g(i_2, j_2)$ as $W^c(i_1, j_1, i_2, j_2)$.

$W^g(i_1, j_1)$ is one of $W^c(i_1, j_1, i_2, j_2)$.

Proof : Because we have Lemma 1, all local paths starting from the outside of the square region to (i_1, j_1) can be calculated by stitching at a certain cell of the square region. And $W^g(i_2, j_2)$ has the best global path in the $i_2 \times j_2$ plane. Thus, the best global path $W^g(i_1, j_1)$ is certainly calculated by the above procedure. □

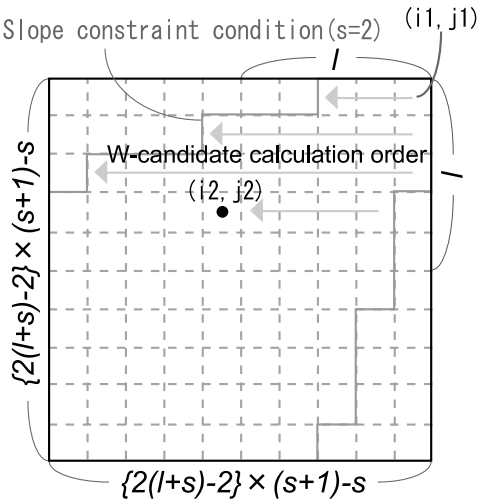


Figure 9. The order of W^l_{temp} and W^c calculation. The slope constraint condition bounds (i_2, j_2). By calculating W^l_{temp} and $W^c(i_1, j_1, i_2, j_2)$in the order indicated by allows, we can acquire all $W^c(i_1, j_1, i_2, j_2)$.

The PS algorithm utilizes Theorem 1. It calculates all candidates $W^c(i_1, j_1, i_2, j_2)$ for a given (i_1, j_1) by the order figure 9 shows, and regards the best one of them as $W^g(i_1, j_1)$. Because we have Theorem 1, it is guaranteed that this $W^g(i_1, j_1)$ is the best global path in the $i_1 \times j_1$ plane.

PS algorithm calculates global paths recurrently from $W^g(1, 2)$ to $W^g(\ \min(I, J+\omega), \min(J, I+\omega)\ )$. Finally, it decides the output global path by comparing values of these global paths. We define $W^g(1, 1) = \emptyset$.

CalGlobalPath(i_1, j_1)
1. **For** all (i_2, j_2)
2. **If** $W^c(i_1, j_1, i_2, j_2)$ is superior than $W^g(i_1, j_1)$
3. Regard $W^c(i_1, j_1, i_2, j_2)$ as new $W^g(i_1, j_1)$
4. **return** $W^g(i_1, j_1)$

PSAlgorithm(A, B)
1. **For** all (i, j) on the $I \times J$ plane
2. $W^g(i, j)$=CalGlobalPath(i, j)
3. **Return** the best global path among $W^g(i', J)$ and $W^g(I, j')$

Table 3. An outline of the PS algorithm.

4. Lower bounding functions

The computational complexity of the PS algorithm is $O(I\omega\{(l+s)s\}^2)$ or $O(J\omega\{(l+s)s\}^2)$. Calculation of $W^c(i_1, j_1, i_2, j_2)$ every (i_1, j_1) takes $O(\{(l+s)s\}^2)$. Although l and s are typically small integer numbers, this is a large time complexity.

Lower bounding functions do not improve the time com-

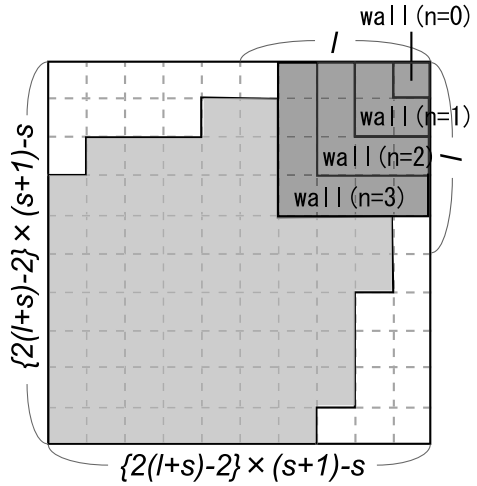


Figure 10. An illustration of a "wall". Path starting from gray cells to the top right corner cell must pass a part of each wall.

plexity, but practically speed up algorithms by pruning obviously futile DTW.

For any sequence pair(A, B), the following inequality must holds:

$$LB(A,B) \leq D_{tw}(A,B) \tag{23}$$

Where $LB(A,B)$ is a lower bounding function. In addition to equation 23, if $\epsilon < LB(A,B)$, the following formula holds:

$$\epsilon < LB(A,B) \leq D_{tw}(A,B) \tag{24}$$

Thus, as we calculate a lower bounding function before DTW execution, we can avoid DTW execution in some cases.

Keogh details existing lower bounding functions for DTW in [7]. We introduce a new lower bounding function suitable for PS algorithm. We represent our proposed function as LB_{morita}.

4.1 The Proposed Lower Bounding Function

Let us consider a square region, $\{2(l+s)-2\}\times(s+1)-s$ on a side, the top right cell is (i_1, j_1), once again. We don't take into consideration cells which are out of the $I \times J$ plane for the following procedure too.

In this square region, we design several "walls" like figure 10 shows. If all $d(a_i, b_j)$ which correspond to cells included in a specific wall are more than the threshold ϵ, there is no local path passing the wall. Thus, if we have the following formula, there is no local path passing $wall_n$:

$$\forall d(a_i, b_j) > \epsilon \mid (i,j) \in wall_n \tag{25}$$

If there is no local path passing a certain wall, there is also no local path starting from the outside of the wall to (i_1, j_1). It is because the PS algorithm employs a monotone increasing function D_{tw-max} as the distance function. We use this mechanism as our proposed lower bounding function. LB_{morita} is defined as follows:

$$LB_{morita} = \min\left(\begin{matrix}\max\begin{pmatrix} b_{j_1-n} - \max(a_{i_1},...,a_{i_1-n}) \\ \min(a_{i_1},...,a_{i_1-n}) - b_{j_1-n}\end{pmatrix} \\ \max\begin{pmatrix} a_{i_1-n} - \max(b_{j_1},...,b_{j_1-n}) \\ \min(b_{j_1},...,b_{j_1-n}) - a_{i_1-n}\end{pmatrix}\end{matrix}\right) \tag{26}$$

n is a given number that holds $0 \leq n < l-1$. The position of a wall is changed by the value of n (see figure 10). If $LB_{morita} > \epsilon$, there is no local path ending at (i_1, j_1). In these cases, we can calculate $W^g(i_1, j_1)$ only to use the next formula:

$$W^g(i_1,j_1) = \min\begin{pmatrix} W^g(i_1-1, j_1) \\ W^g(i_1, j_1-1) \\ W^g(i_1-1, j_1-1)\end{pmatrix} \tag{27}$$

Constant numbers needed for LB_{morita} calculation can be achieved by a scanning of A and B. Thus, the computational complexity of the PS algorithm using LB_{morita} is $O(I\omega\{(l+s)s\}^2)$ or $O(J\omega\{(l+s)s\}^2)$. This is same as the case that we don't use lower bounding functions. However, lower bounding functions would practically speed up the over all calculation procedure.

PSAlgorithmUsingLBmorita(A,B)
1. **For** all (i,j) on the $I \times J$ plane
2. **If** $LB_{morita}(i,j,n) > \epsilon$
3. $W^g(i,j)$ is calculated by Equ.27
4. **Else**
2. $W^g(i,j)$=CalGlobalPath(i,j)
6. **Return** the best global path among $W^g(i',J)$ and $W^g(I,j')$

Table 4. An outline of the PS algorithm using our proposed lower bounding function LB_{morita}.

5. Experimental Results

In this section we apply the PS algorithm to an eye movement data analysis. It is well known that we look at attractive objects by the center of the retina. In these cases, the eye position's trajectory and the attended object position's trajectory become similar. Therefore, researchers can estimate what a subject pays attention to by an eye movement analysis. However, when an attended object is moving, the eye position often delays from the object position[4], and this delay time varies depending on the nature of the subject and object. In addition, we do not know when and how long subjects track objects by eye movements before analysis. The PS algorithm is very useful for analyzing the data having these features.

We acquired eye movement data from four undergraduate students at the Keio University. Subjects watched a dance video after being instructed to attend to a specific part of the dancer's body. We measured their eye movements while they watch the video.

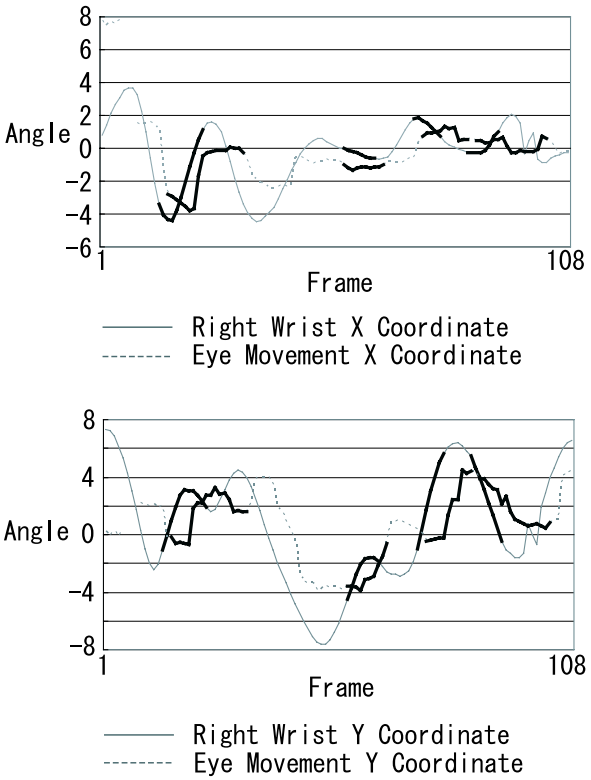


Figure 11. Results of a trial. Similar subsequences are drawn by bold lines There are some missing data (perhaps caused by blinking). Note that we calculate distances between elements as the 2D Euclidean distance.

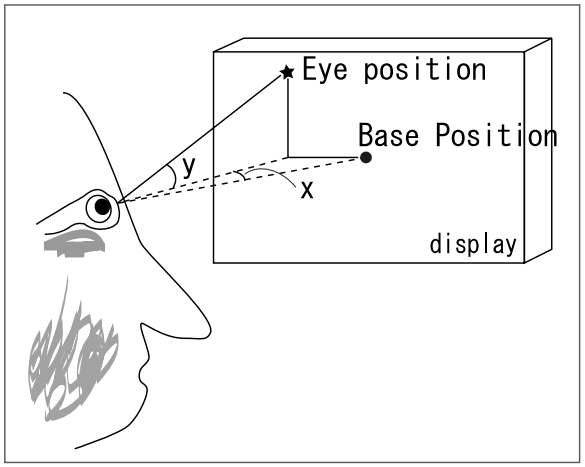


Figure 12. In our experiments, we measured two-dimensional eye rotation angle.

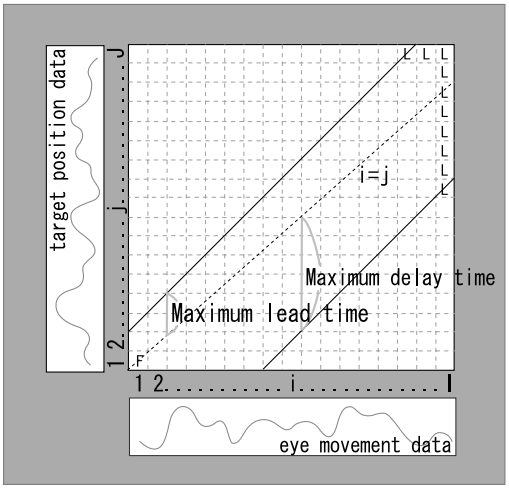


Figure 13. Modifying the global constraint condition. By this modification, the PS algorithm becomes to output the best global path that starts from F to L.

After the measurement, we executed the PS algorithm for pairs of eye movement data and several body parts position data. Then we reaffirmed that the amount of the tracking time for the part to which we had instructed to attend is much larger than that for other body parts. These results are consistent with previous works[9].

Figure 11 illustrates the eye movement data and the right wrist position data at a specific trial in which the subject was required to attend to the right wrist. Each data consists of two-dimensional eye rotation angle$< x, y >$, horizontal angle x and vertical angle y (figure 12). Figure 11 also shows similar subsequences by bold lines. Note that we use the two-dimensional Euclidian distance for similarity judging.

We interpreted the warping window width as a maximum lead and delay time of eye movements against the object (figure 13). The threshold was set at 1.5 degrees, the maximum delay time of eye movements was set at 0.4 seconds, the maximum lead time of eye movements was set at 0 second and the lower bound of subsequence length was set at 0.167 seconds (5 frames). The sampling rate of both sets of data is 30 frames per second.

6. Summary and Conclusion

In this paper, we proposed a novel algorithm, called Paths Stitching, to extract time-ordered pairs of similar subsequences of variable length. This algorithm is carried out by stitching local paths one after another. Because local paths are obtained by DTW, the PS algorithm is robust for a locally fluctuant time-difference between subsequences. We also introduced a new lower bounding function, LB_{morita}, suitable for the PS algorithm to speed up it. Finally we applied the PS algorithm to an eye movement analysis to demonstrate its feasibility.

Here we again compare the PS algorithm with the popular technique that Agrawal et al. proposed[2]. In the matching phase, their popular technique allows roughly linear scale-normalization, offsetting translation, offsetting a global time-difference and removing outliers. On the other hand, the PS algorithm allows nonlinear time-normalization and offsetting a global time-difference. Our proposed algorithm holds the upper bound of the value of a offset global time-difference by the global constraint condition, although Agrawal's algorithm does not restrict it. Both of them extract the best sequence of pairs of subsequences that are similar under their own measure.

The above features of the PS algorithm are almost attributable to DTW. Thus, the domain in which the PS algorithm is useful may be similar to the domain in which DTW is useful. For example, like DTW, the PS algorithm is effective for time sequences of different sampling rates. In these cases, we may correctly deal with the pair of sequences by modifying the global constraint condition and the slope constraint condition.

The problem of extracting pairs of similar subsequences of variable length has not been paid a big attention. But we think there are many unfamiliar domains in which our PS algorithm is very effective as preprocessing, like an eye movement analysis. It is our future work to find these domains and fully develop our approach.

7. REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Proceedings of the 4th International Conference of Foundations of Data Organization and Algorithms (FODO)*, pages 69 – 84, 1993.

[2] R. Agrawal, K. Lin, H. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceedings of the 21st International Confefence on Very Large Databases*, pages 490–501, 1995.

[3] D. Berndt and J. Clifford. Finding patterns in time series: A dynamic programming approach. In *Advances Knowledge Discovery Data Mining*, pages 229–248. AAAI/MIT, 1996.

[4] H. Collewijn and E.P.Tamminga. Human smooth and saccadic eye movements during voluntary pursuit of different target motions on different backgrounds. *J.Physiology*, 351:217–250, 1984.

[5] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 419–429, 1994.

[6] T. Kahveci and A. K. Singh. Variable length queries for time series data. In *Proceedings of the 17th International Conference on Data Engineering*, pages 273 – 282, 2001.

[7] E. Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 406–417, 2002.

[8] E. J. Keogh. and M. J. Pazzani. An indexing scheme for fast similarity search in large time series databases. In *Proceedings of the 11th International Conference on Scientific and Statistical Database Management*, pages 56–67, 1999.

[9] B. Khurana and E. Kowler. Shared attentional control of smooth eye movements and perception. *Vision Research*, 27(9):1603–1618, 1987.

[10] S. Kim, S. Park, and W. W. Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In *Proceedings of the 17th International Conference on Data Engineering*, pages 607–614, 2001.

[11] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, And Signal Processing,*, ASSP-26(1):43–49, 1978.

[12] B. Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings of the 14th International Conference on Data Engineering*, pages 201–208, 1998.

Everything you know about Dynamic Time Warping is Wrong

Chotirat Ann Ratanamahatana Eamonn Keogh

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

{ ratana, eamonn }@cs.ucr.edu

ABSTRACT

The Dynamic Time Warping (DTW) distance measure is a technique that has long been known in speech recognition community. It allows a non-linear mapping of one signal to another by minimizing the distance between the two. A decade ago, DTW was introduced into Data Mining community as a utility for various tasks for time series problems including classification, clustering, and anomaly detection. The technique has flourished, particularly in the last three years, and has been applied to a variety of problems in various disciplines.

In spite of DTW's great success, there are still several persistent "myths" about it. These myths have caused confusion and led to much wasted research effort. In this work, we will dispel these myths with the most comprehensive set of time series experiments ever conducted.

Keywords

Dynamic Time Warping. Data Mining. Experimentation.

1. INTRODUCTION

In recent years, classification, clustering, and indexing of time series data have become a topic of great interest within the database/data mining community. The Euclidean distance metric has been widely used [17], in spite of its known weakness of sensitivity to distortion in time axis [15]. A decade ago, the Dynamic Time Warping (DTW) distance measure was introduced to the data mining community as a solution to this particular weakness of Euclidean distance metric [3]. This method's flexibility allows two time series that are similar but locally out of phase to align in a non-linear manner. In spite of its $O(n^2)$ time complexity, DTW is the best solution known for time series problems in a variety of domains, including bioinformatics [1], medicine [5], engineering, entertainment [30], etc.

The steady flow of research papers on data mining with DTW became a torrent after it was shown that a simple lower bound allowed DTW to be indexed with no false dismissals [15]. The lower bound requires that the two sequences being compared are of the same length, and that the amount of warping is constrained. This work allowed practical applications of DTW, including real-time query-by-humming systems [30], indexing of historical handwriting archives [24], and indexing of motion capture data [6].

In spite of the great success of DTW in a variety of domains, there still are several persistent myths about it. These myths have caused great confusion in the literature, and led to the publication of papers that solve apparent problems that do not actually exist. The three major myths are:

Myth 1: *The ability of DTW to handle sequences of different lengths is a great advantage, and therefore the simple lower bound that requires different-length sequences to be reinterpolated to equal length is of limited utility* [18][27][28]. In fact, as we will show, there is no evidence in the literature to suggest this, and extensive empirical evidence presented here suggests that comparing sequences of different lengths and reinterpolating them to equal length produce no statistically significant difference in accuracy or precision/recall.

Myth 2: *Constraining the warping paths is a necessary evil that we inherited from the speech processing community to make DTW tractable, and that we should find ways to speed up DTW with no (or larger) constraints*[27]. In fact, the *opposite* is true. As we will show, the 10% constraint on warping inherited blindly from the speech processing community is actually too large for real world data mining.

Myth 3: *There is a need (and room) for improvements in the speed of DTW for data mining applications*. In fact, as we will show here, if we use a simple lower bounding technique, DTW is essentially $O(n)$ for data mining applications. At least for CPU time, we are almost certainly at the asymptotic limit for speeding up DTW.

In this paper, we dispel these DTW myths above by empirically demonstrate our findings with a comprehensive set of experiments. In terms of number of objective datasets and size of datasets, our experiments are orders of magnitude greater than anything else in the literature. In

particular, our experiments required more than eight billion DTW comparisons.

Before beginning our deconstruction of these myths, it would be remiss of us not to note that several early papers by the second author are guilty of echoing them. This work is part of an effort to redress these mistakes. Likewise, we have taken advantage of the informal nature of a workshop to choose a tongue-in-cheek attention grabbing title. We do not really mean to imply that the entire community is ignorant of the intricacies of DTW.

The rest of the paper is organized as follows. In Section 2, we give an overview of Dynamic Time Warping (DTW) and its related work. The next three sections consider each of the three myths above. Section 6 suggests some avenues for future researches, and Section 7 gives conclusions and directions for future work. Because we are testing on a wide range of real and synthetic datasets, we have placed the details about them in Appendix A to enhance the flow of the paper.

2. BACKGROUND AND RELATED WORK

The measurement of similarity between two time series is an important subroutine in many data mining applications, including classification [11][14], clustering [1][10], anomaly detection [9], rule discovery [8], and motif discovery [7]. The superiority of DTW over Euclidean distance metric for these tasks has been demonstrated by many authors [1][2][5][29]. We will first begin with a review of some background material on DTW and its recent extensions, which contributes to our main motivation of this paper.

2.1 REVIEW OF DTW

Suppose we have two time series, a sequence Q of length n, and a sequence C of length m, where

$$Q = q_1,q_2,\ldots,q_i,\ldots,q_n \quad (1)$$

$$C = c_1,c_2,\ldots,c_j,\ldots c_m \quad (2)$$

To align these two sequences using DTW, we first construct an n-by-m matrix where the (i^{th}, j^{th}) element of the matrix corresponds to the squared distance, $d(q_i, c_j) = (q_i - c_j)^2$, which is the alignment between points q_i and c_j. To find the best match between these two sequences, we retrieve a path through the matrix that minimizes the total cumulative distance between them as illustrated in Figure 1. In particular, the optimal path is the path that minimizes the warping cost

$$DTW(Q,C) = \min\left\{\sqrt{\sum_{k=1}^{K} w_k}\right\} \quad (3)$$

where w_k is the matrix element $(i,j)_k$ that also belongs to k^{th} element of a warping path W, a contiguous set of matrix elements that represent a mapping between Q and C.

This warping path can be found using dynamic programming to evaluate the following recurrence.

$$\gamma(i,j) = d(q_i,c_j) + \min\{\gamma(i-1,j-1), \gamma(i-1,j), \gamma(i,j-1)\} \quad (4)$$

where $d(i,j)$ is the distance found in the current cell, and $\gamma(i,j)$ is the cumulative distance of $d(i,j)$ and the minimum cumulative distances from the three adjacent cells.

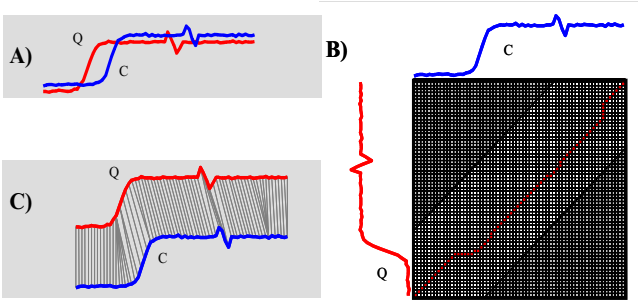


Figure 1. A) Two similar sequences *Q* and *C*, but out of phase. B) To align the sequences, we construct a warping matrix and search for the optimal warping path, shown with solid squares. Note that the 'corners' of the matrix (shown in dark gray) are excluded from the search path as part of an Adjustment Window condition. C) The resulting alignment.

To reduce the number of paths to consider during the computation, several well-known constraints (*Boundary Conditions, Continuity condition, Monotonic condition, and Adjustment Window Condition)* have been applied to the problem to restrict the moves that can be made from any point in the path and so restrict the number of paths that need to be considered. Figure 1 B) illustrates a particular example of the Adjustment Window Condition (or Warping Window Constraints) with the Sakoe-Chiba Band [26]. The width of this constraint is often set to 10% of the length of the time series [1][22][26].

2.2 LOWER BOUNDING THE DTW DISTANCE

A recent extension to DTW that significantly speeds up the DTW calculation is a lower bounding technique based on the warping window (envelope) [15].

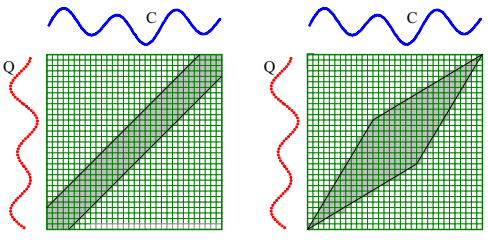


Figure 2. The two most common constraints in the literature are the Sakoe-Chiba Band and the Itakura Parallelogram

Figure 2 illustrates two of the most frequently used global constraints in the literature, the Sakoe-Chiba Band [26] and

the Itakura Parallelogram [13]. The latter is widely used in the speech community.

The lower bound is only defined for sequences of the same length; if the sequences are of different lengths, one of them must be reinterpolated. This lower bounding technique uses the warping window to create a bounding envelope above and below the query sequence. Then the squared sum of the distances from every part of the candidate sequence not falling within the bounding envelope, to the nearest orthogonal edge of the bounding envelope, is returned as its lower bound. The technique is illustrated in Figure 3.

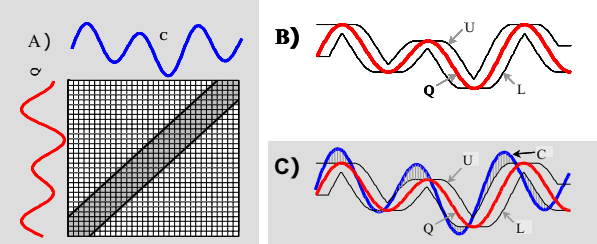


Figure 3. The Sakoe-Chiba Band A) can be used to create an envelope B) around a query sequence *Q*. The Euclidean distance between any candidate sequence *C* and the closest external part of the envelope C) is a lower bound for the DTW distance

For clarity, in Table 1, we will show a trivial algorithm that can exploit *any* lower bound to do faster sequential search. This algorithm is taken from Table 2 of [15].

Table 1. An algorithm that uses a lower bounding distance measure to speed up the sequential scan search for the query Q

```
Algorithm Lower_Bounding_Sequential_Scan(Q)
1.  best_so_far = infinity;
2.  for all sequences in database
3.   LB_dist = lower_bound_distance(Ci, Q);
4.      if LB_dist < best_so_far
5.          true_dist = DTW(Ci, Q);
6.          if true_dist < best_so_far
7.              best_so_far = true_dist;
8.              index_of_best_match = i;
9.          endif
10.     endif
11. endfor
```

Note that the tightness of these lower bounds and pruning power essentially depend on the size of the warping window used as well. In general, the smaller the area of allowed warping, the more we can take advantage of pruning. As noted above, the best size of this warping is subject to controversy; we will examine the question in Section 4.

For clarity, we will summarize before continuing. If we are willing to force the sequences to be of the same length, and to constrain the warping, then we have a simple solution for speeding up similarity search under DTW. We will call this solution **4S** (Simple Straw man for Similarity Search). As we shall see, the papers that try to speed up this simple approach, or relax its two assumptions, are motivated and misled by the myths discussed above.

3. DOES COMPARING SEQUENCES OF DIFFERENT LENGTHS HELP OR HURT?

Many recent papers suggest that the ability of classic DTW to deal directly with sequences of different length is a great advantage; some paper titles even contain the phrase *"…of different lengths"* [4][21] showing their great concerns in solving this issue. As further examples, consider the following quotes taken from recent papers: "*Time warping enables sequences with similar patterns to be found even when they are of different lengths*" [18], or " *(DTW is) a more robust distance measure than Euclidean distance in many situations, where sequences may have different lengths*" [28] or *"(DTW) can be used to measure similarity between sequences of different lengths*". Some of these papers further suggest that the simple **4S** solution to DTW similarity search is not useful because it requires that sequences of different lengths to be reinterplolated to the same length, and use this fact to motive new approaches: for example "***(4S)** only works when the data and query sequences are of the same length."* [27].

These claims are surprising in that they are not supported by any empirical results in the papers in question. Furthermore, an extensive literature search through more than 500 papers dating back to the 1960's failed to produce any theoretical or empirical results to suggest that simply making the sequences to be of the same length has any detrimental effect.

To test our claimed hypothesis that there is no significant difference in accuracies between using variable-length time series and equal-length time series in DTW calculation, we carry out an experiment as follows.

For all variable-length time series datasets (Face, Leaf, Trace, and Wordspotting – See Appendix A for dataset details), we compute 1-nearest-neightbor classification accuracies (leaving-one-out) using DTW for all warping window sizes (1% to 100%) in two different ways:

1) The **4S** way; we simply reinterpolated the sequences to have the same length.
2) By comparing the sequences directly using their original lengths.

The latter case is not as simple as one might think since we need to normalize the returned distance in some way. All things being equal, we would expect longer sequences to be further apart than short sequences, since they have more dimensions on which to accumulate noise. The following normalization policies have appeared in the literature or are common sense ideas.

- No normalization on the distance.
- Normalize by the length of the optimal warping path.
- Normalize by the length of the shorter time series (for each pair of the time series during each DTW computation).
- Normalized by the length of the longer time series.

To give the benefit of the doubt to different-length case, for each warping window size, we do all four possible normalizations above, and the best performing of the four options is recorded as the accuracy for the variable-length DTW calculation.

For completeness, we test over every possible warping constraint size. Note that we start the warping window size of 1% instead of 0% since 0% size is Euclidean distance metric, which is undefined when the time series are not of the same length. Also, when measuring the DTW distance between two time series of different lengths, the percentage of warping window applied is based on the length of the longer time series to ensure that we allow adequate amount of warping for each pair and deliver a fair comparison.

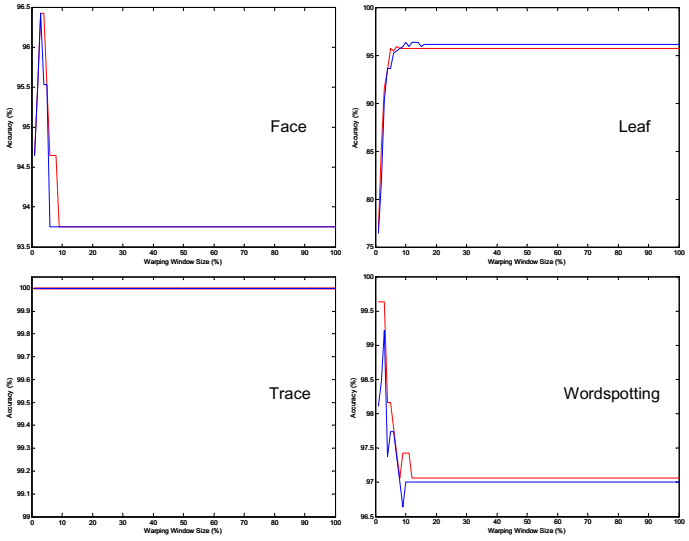


Figure 4. A comparison of the classification accuracies between variable-length datasets (dotted lines) and the (reinterpolated) equal-length datasets (solid lines). The two options produce such similar results that in many places the lines overlap.

The variable-length datasets are then linearly reinterpolated to have the same length of the longest time series within each dataset. After that, we simply compute the classification accuracies using DTW for all warping window sizes (1% to 100%) for each dataset. The results are shown in Figure 4.

Note that the experiments do strongly suggest that changing the amount of warping allowed does affect the accuracy (an issue that will be discussed in depth in the next section), but over the entire range on possible warping widths, the two approaches are nearly indistinguishable. Furthermore, a two-tailed test using a significance level of 0.05 between each variable-length and equal-length pair indicates that there is no statistically significant difference between the accuracy of the two sets of experiments. An even more telling result is the following. In spite of extensive experience with DTW and an extensive effort, we were unable to create an artificial problem where reinterpolating made a significant difference in accuracy. To further reinforce our claim, we also reinterpolate the datasets to have the equal length of the shortest and averaged length of all time series within the dataset. We still achieve similar findings.

These results strongly suggest that work allowing DTW to support similarity search that does require reinterpolation, is simply solving a problem that does not exist. Subsequently, while Wong and Wong claimed, "*(DTW is useful) to measure similarity between sequences of different lengths*" [28] we must recall that two *Wongs* do not make a right[1]. The often-quoted utility of DTW being able to support the comparison of sequences of different lengths is simply a myth.

4. ARE NARROW CONSTRAINTS BAD?

Apart from (slightly) speeding up the computation, warping window constraints were originally applied mainly to prevent pathological warping (where a relatively small section of one sequence maps to a much larger section of another). The vast majority of the data mining researchers have used a Sakoe-Chiba Band with a 10% width for the global constraint [1][22][26]. This setting seems to be the result of historical inertia, inherited from the speech processing community, rather than some remarkable property of this particular constraint.

Some researchers believe that having wider warping window contributes to improvement in accuracy [30]. Or without realizing the great effect of the warping window size on accuracies, some applied DTW with no warping window constraints [20], or did not specify the window size used in the experiments [19] (the latter case makes it particularly difficult for others to reproduce the experiment results). In [27], the authors bemoan the fact that *"(4S) cannot be applied when the warping path is not constrained*" and use

[1] Yes, this is a very poor joke!

this fact to justify introducing an alterative approach that works for the unconstrained case.

To test the effect of the warping window size to the classification accuracies, we performed an empirical experiment on all seven classification datasets. We vary the warping window size from 0% (Euclidean) to 100% (no constraint/full calculation) and record the accuracies.

Since we have shown in Section 3 that reinterpolation of time series into the same length is at least as good as (or better than) using the original variable-length time series, we linearly interpolate all variable-length datasets to have the same length of the longest time series within the dataset and measure the accuracy using the 1-nearest-neighbor with leaving-one-out classification method. The results are shown in Figure 5.

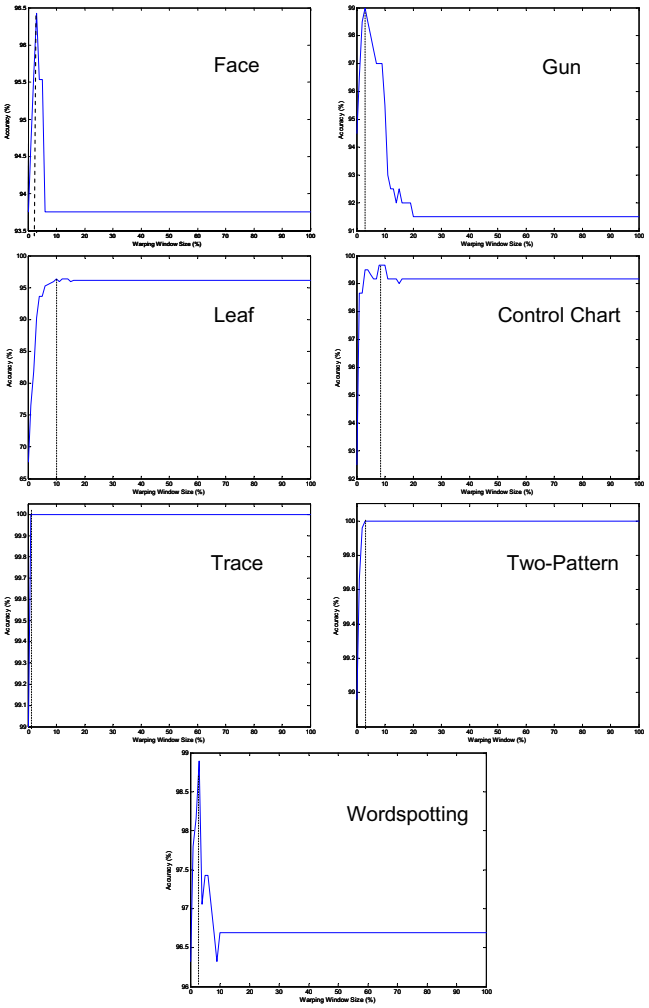


Figure 5. The classification accuracies for all warping window sizes. All accuracies peak at very small window sizes.

As we hypothesized, wider warping constraints do not always improve the accuracy, as commonly believed [30]. More often, the accuracy peaks very early at much smaller window size, as shown in Table 2 below. In essence, the results can be summarized by noting that a little warping is a good thing, but too much warping is a bad thing.

Table 2. The warping window size that yields maximum classification accuracy for each dataset, using DTW with Sakoe-Chiba Band.

Dataset	Max Accuracy (%)	Warping Window Size (%)
Face	96.43	3
Gun	99.00	3
Leaf	96.38	10
Syn_contrl_chrt	99.67	8
Trace	100.00	1
TwoPatterns	100.00	3
Wordspotting	98.90	3

We also did an additional experiment, where half of the objects in the databases are randomly removed from the database in each iteration. We measure the classification accuracies for each database size, as shown in Figure 6. As the database size decreases, the classification accuracy also declines and the peak appears at larger warping window size.

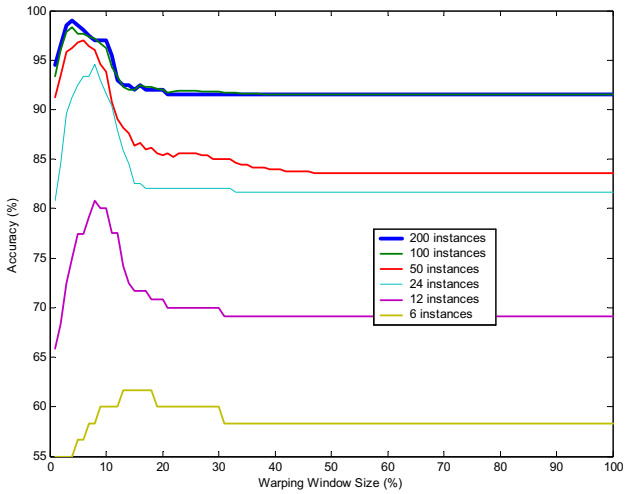


Figure 6. With fewer objects in the databases, the accuracies become less accurate and peak at larger window size

This finding suggests that warping window size adjustment does affect accuracy, and that the effect also depends on the database size. This in turn suggests that we should find the best warping window size on realistic (for the task at hand) database sizes, and not try to generalize from toy problems.

To summarize, there is no evidence to support the idea that we need to be able to support wider constraints. While it is possible that there exist some datasets somewhere that

could benefit from wider constraints, we found no evidence for this in a survey of more than 500 papers on the topic. More tellingly, in spite of extensive efforts, we could not even create a large synthetic dataset for classification that needs more than 10% warping.

All the evidence suggests that narrow constraints are *necessary* for accurate DTW, and the "need" to support wide (or no) constraints is just a myth.

5. CAN DTW BE FURTHER SPEEDED UP?

Smaller warping windows speed up the DTW calculations simply because there is less area of the warping matrix to be searched. Prior to the introduction of lower bounding, the amount of speedup was directly proportional to the width of the warping window. For example, a nearest neighbor search with a 10% warping constraint was almost exactly twice as fast as a search done with a 20% window. However, it is important to note that with the introduction of lower bounding based on warping constraints (i.e. **4S**), the speedup is now highly nonlinear in the size of the warping window. For example, a nearest neighbor search with a 10% warping constraint may be many times faster than twice a search done with a 20% window.

In spite of this, many recent papers still claim that there is a need and room for further improvement in speeding up DTW. For example, a recent paper suggested "*dynamic time warping incurs a heavy CPU cost…*" Surprisingly, as we will now show, the amortized CPU cost of DTW is essentially O(n) if we use the trivial **4S** technique.

To really understand what is going on, we will avoid measuring the efficiency of DTW when using index structures. The use of such index structures opens the possibility of implementation bias [17]; it is simply difficult to know if the claimed speedup truly reflects a clever algorithm, or simply the care in choice of buffer size, caching policy, etc.

Instead, we measure the computation time of DTW for each pair of time series in terms of the amortized percentage of the warping matrix that needs to be visited for each pair of sequences in our database. This number depends only on the data itself and the usefulness of the lower bound. As a concrete example, if we are doing a one nearest neighbor search on 120 objects with a 10% warping window size, and the **4S** algorithm only needs to examine 14 sequences (pruning the rest), then the amortized cost for this calculation would be (w * 14) / 120 = 0.12*w, where w is the area (in percentage) inside the warping window constraint along the diagonal (Sakoe-Chiba band). Note that 10% warping window size does not always occupy 10% of the warping matrix; it mainly depends on the length of the sequence as well (longer sequences give smaller w). In contrast, if **4S** was able to prune all but 3 objects, the amortized cost would be (w * 3) / 120 = 0.03*w.

The amount of pruning we should actually expect depends on the lower bounds. For example, if we used a trivial lower bound hard-coded to zero (pointless, but perfectly legal), then line 4 of Table 1 would always be true, and we would have to do DTW for every pair of sequences in our dataset. In this case, amortized percentage of the warping matrix that needs to be accessed for each sequence in our database would exactly be the area inside the warping window. If, on the other hand, we had a "magic" lower bound that returned the true DTW distance minus some tiny epsilon, then line 4 of the Table 1 would rarely be true, and we would have to do the full DTW calculation only rarely. In this case, the amortized percentage of the warping matrix that needs to be accessed would be very close to zero.

We measured the amortized cost for all our datasets, and for every possible warping window size. The results are shown in Figure 7. Figure 8 shows the zoom-in of the results from 0 to 10 % warping window size

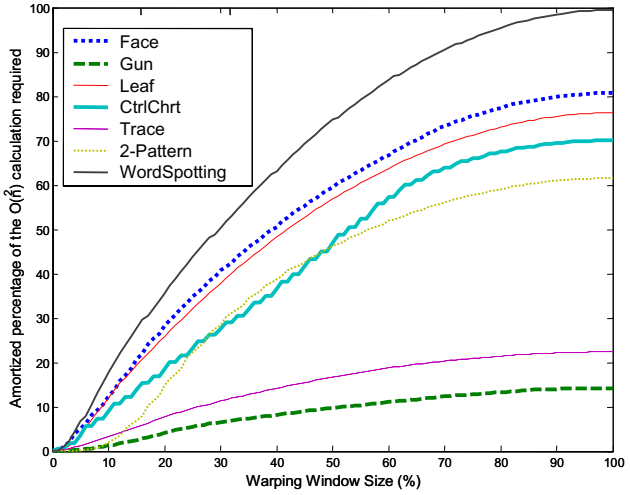


Figure 7. The amortized percentage of warping matrix that needs to be accessed during the DTW calculation for each warping window size. The use of a lower bound helps prune off numerous unnecessary calculations.

The results are very surprising. For reasonably large datasets, simply using a good lower bound insures that we rarely have to use the full DTW calculation. In essence, we can say that DTW is effectively O(n), and not O(n^2), when searching large datasets.

For example, in the Gun, Trace, and 2-Pattern problems (all maximum accuracy at 3% warping), we only need to do much less than half a percent of the O(n^2) work that we would have been forced to do without lower bounding. For some of the other datasets, it may appear that we need to do a significant percentage of the CPU work. However, as we will see below, these results are pessimistic in that they reflect the small size of these datasets.

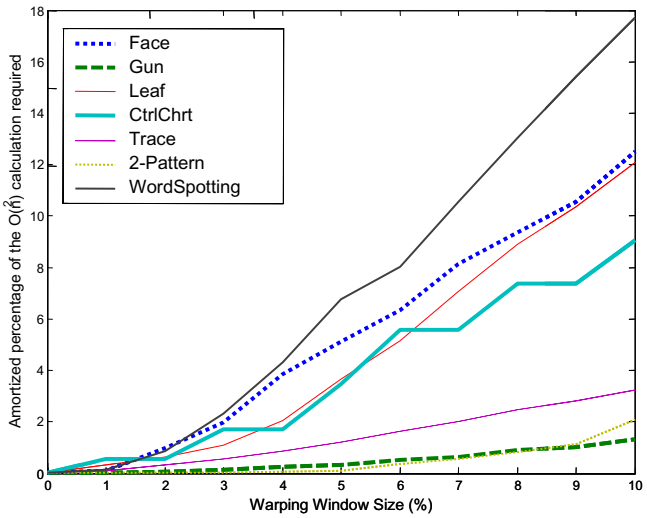


Figure 8. Zoom-in of Figure 6, from 0 to 10% warping window size. Note that from Section 4, we recognize that the most accurate results for all datasets happen in this range.

If the amortized cost of DTW is linear, where does the claimed improvement from recent papers come from? It is true that these approaches typically use indexes, rather than sequential search, but an index must do costly random access rather than the optimized linear scans of sequential search. In order to simply break even in terms of disk access time, they must avoid looking at more than 10% of the data [16], but for time series where even the reduced dimensionality (i.e. the Fourier or wavelet coefficients) is usually greater than 20 [17], it is not obvious that this is possible.

Some recent papers that claim speedups credit the improved lower bounds, for example "*…we present progressively tighter lower bounds… that allow our method to outperform (4S)* " [27]. Indeed, it might be imagined that speedup could be obtained by having tighter lower bounds. Surprisingly, this is not true! We can see this with the following simple experiment. Let us imagine that we have a wonderful lower bound, which always returns a value that is within 1% of the correct value (more concretely, a value uniformly distributed between 99% and 100% of the true DTW value). We will call this idealized lower bound *LB_Magic*. In contrast, the current best-known lower bounds typically return a value between 40% and 60% of the true value [15].

We can compare the speedup obtained by LB_Magic with the current best lower bound, LB_Keogh [15], on 1-nearest neighbor search. Note that we have to cheat for LB_Magic by doing the full DTW calculation then assigning it a value up to 1% smaller. We will use a warping constraint of 5%, which is about the mean value for the best accuracy (cf. Section 4). As before, we measured the amortized percentage of the warping matrix that needs to be accessed for each sequence in our database. For this experiment, we use a *randomwalk* data of length 128 data points, and vary the database size from 10 objects to 40,960 objects. Figure 9 shows the results.

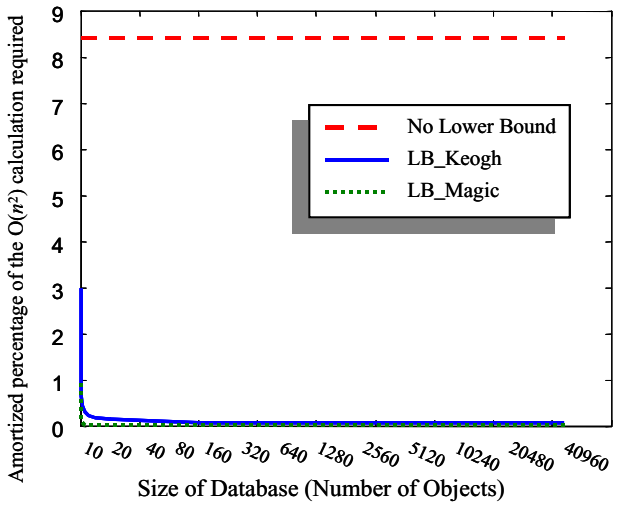


Figure 9. Amortized percentage of the warping matrix that needs to be accessed. As the size of the database is increasing, the amortized percentage of the warping matrix accessed becomes closer to zero.

Once again, the results are very surprising. The idealized LB_Magic allows a very impressive speedup; for the largest size database, it eliminates 99.997% of the CPU effort. However, the very simple lower bounding technique that has been in the literature for several years is able to eliminate 99.369% of the CPU effort! The difference is not quite so dramatic for very small datasets, say less than 160 objects. But here we can do unoptimized search in much less than a hundredth of a second. Note that we obtain similar results for other datasets.

To summarize, for problems involving a few thousand sequences or more, each with a few hundred data points, the "*significant CPU cost of DTW*" is simply non-issue (as for problems involving less than a few thousand sequences, we can do them in less than a second anyway).

The lesson for the data mining community from this experiment is the following; it is almost certainly pointless to attempt to speed up the CPU time for DTW by producing tighter lower bounds. Even if you could produce a magical lower bound, the difference it would make would be tiny, and completely dwarfed by minor implementation choices.

6. AVENUES FOR FUTURE RESEARCH

In this section, we will attempt to redress the somewhat negative tone of this paper by suggesting many avenues for future research with DTW. Since it has been demonstrated by many authors that DTW is the solution to many data mining problems, we would like to present some of the

other applications or problems that can effectively benefit from DTW distance measure.

6.1 VIDEO RETRIEVAL

Generally, research on content-based video retrieval represents the content of the video as a set of frames, leaving out the temporal features of frames in the shot. However, for some domains, including motion capture editing, gait analysis, and video surveillance, it may be fruitful to extract time series from the video, and index *just* the time series (with pointers back to the original video). Figure 10 shows an example of a video sequence that is transformed into a time series. This example is the basis for the Gun dataset discussed in Appendix A.

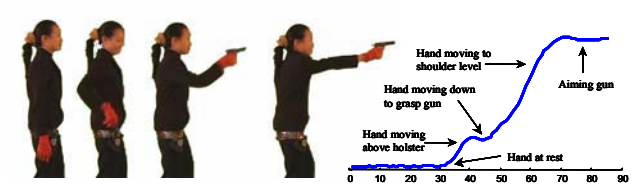


Figure 10. Stills from a video sequence; the right hand is tracked, and converted into a time series

One obvious reason why using time series representation may be superior to working with the original data is the massive reduction in dimensionality, which enhances the ease of storage, transmission, analysis, and indexing. Moreover, it is much easier to make the time series representation invariant to distortions in the data, such as time scaling and time warping.

6.2 IMAGE RETRIEVAL

For some specialized domains, it can be useful to convert the images into "pseudo time series". For example, consider Figure 11 Below. Here, we have converted an image of a leaf into a time series by measuring the local angle of a trace of its perimeter. The utility of such a transform is similar to that for video retrieval.

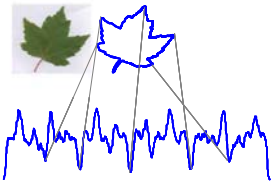


Figure 11. An example of a leaf image converted into a "pseudo time series"

6.3 HANDWRITING RETRIEVAL

The problem of transcribing and indexing existing historical archives is still a challenge. For even such a major historical figure as Isaac Newton, there exists a body of unpublished, handwritten work exceeding one million words. For other historical figures, there are even larger collections of handwritten text. Such collections are potential goldmines for researchers and biographers. Recent work by [24] suggests that DTW may be best solution to this problem.

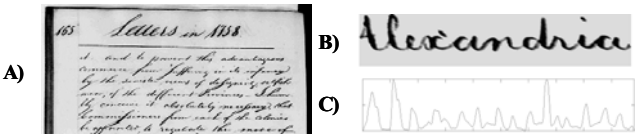


Figure 12. A) An example of handwritten text by George Washington. B) A zoom-in on the word "Alexandria", after being processed to remove slant. C) Many techniques exist to convert 2-D handwriting into a time series; in this case, the projection profile is used (Fig. created by R. Manmatha).

6.4 TEXT MINING

Surprisingly, we can also transform text into a time series representation. For instance, we consider a problem of translating biblical text in two different languages (English and Spanish). The bible text is converted into bit streams according to the occurrences of the chosen word in the text. For example, subsection of the bible containing the word 'God' in "In the beginning God created the heaven and the earth" will be represented by "0001000000". Then the bit streams are converted into time series by recording the number of word occurrences within the predefined sliding window.

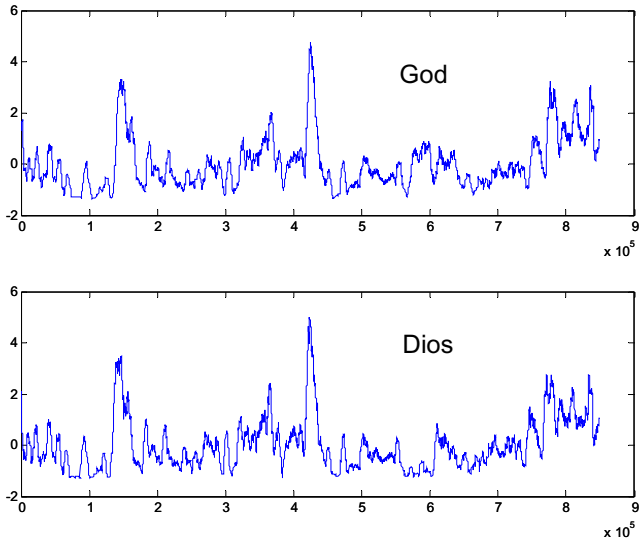


Figure 13. Times series of the number of occurrences of the word 'God' in English (top) and 'Dios' in Spanish (bottom) bible text using 6,000 words as the window size (z-normalized and reinterpolated to the same length). The two time series are almost identical.

The intuition behind this approach is that for each appearance of each word in English, there must be a corresponding Spanish word that also appears in the same vicinity in the Spanish bible text. However, there can be some discrepancies in the number of words in the entire text as well as the position of the word within the sentence

between the two languages. This can be overcome by DTW technique.

These suggestions are only subsets of variety of problems that could benefit from transforming them into time series, which DTW could be trivially applied to. There are still considerable amount of applications that are left to be explored.

7. CONCLUSIONS AND FUTURE WORK

In this work, we have pointed out and investigated some of the myths in Dynamic Time Warping measure. We empirically validated our three claims.

We hope that our results will help researchers focus on more useful problems. For example, while there have been dozens of papers on speeding up DTW in the last decade, there has only been one on making it more accurate [23]. Likewise, we feel that the speed and accuracy of DTW that we have demonstrated in this work may encourage researchers to apply DTW to a wealth of new problems/domains.

8. ACKNOWLEDGMENTS

All datasets used in this paper are available upon request, by emailing either author. We thank Yasushi Sakurai for his useful comments. We note that some of the claims in this paper might be controversial. We welcome comments and criticism, and will be happy to run experiments on your favorite dataset. The online version on this paper (at www.cs.ucr.edu/~eamonn/selected_publications.php) will be updated within 48 hours of any contradictory evidence being found.

9. REFERENCES

[1] Aach, J. & Church, G. (2001). Aligning gene expression time series with time warping algorithms. Bioinformatics. Vol. 17, pp. 495-508.

[2] Bar-Joseph, Z., Gerber, G., Gifford, D., Jaakkola, T. & Simon, I. (2002). A new approach to analyzing gene expression time series data. In Proceedings of the 6th Annual International Conference on Research in Computational Molecular Biology, pp. 39-48.

[3] Berndt, D. and Clifford, J. (1994). Using dynamic time warping to find patterns in time series. AAAI Workshop on Knowledge Discovery in Databases, pp. 229-248.

[4] Bozkaya, T, Yazdatani, Z, and Ozsoyoglu, Z.M. (1997). Matching and Indexing Sequences of Different Lengths. CIKM-97.

[5] Caiani, E.G., Porta, A., Baselli, G., Turiel, M., Muzzupappa, S., Pieruzzi, F., Crema, C., Malliani, A., & Cerutti, S. (1998). Warped-average template technique to track on a cycle-by-cycle basis the cardiac filling phases on left ventricular volume. IEEE Computers in Cardiology, pp. 73-76.

[6] Cardle, M. (2003). Music-Driven Animation. Ph.D. Thesis, Cambridge University.

[7] Chiu, B. Keogh, E., & Lonardi, S. (2003). Probabilistic Discovery of Time Series Motifs. In the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. August 24-27, 2003. Washington, DC, USA.

[8] Das, G., Lin, K., Mannila, H., Renganathan, G. & Smyth, P. (1998). Rule discovery from time series. Proceedings of the 4th International Conference of Knowledge Discovery and Data Mining, pp. 16-22, AAAI Press.

[9] Dasgupta, D. & Forest, S. (1999). Novelty Detection in Time Series Data using Ideas from Immunology. In Proceedings of the International Conference on Intelligent Systems.

[10] Debregeas, A. & Hebrail, G. (1998). Interactive interpretation of Kohonen maps applied to curves. Proceedings of the 4th International Conference of Knowledge Discovery and Data Mining, pp. 179-183.

[11] Diez, J.J.R. & Gonzalez, C.A. (2000). Applying boosting to similarity literals for time series Classification. Multiple Classifier Systems, 1st International Workshop. pp. 210-219.

[12] Geurts, P. (2002). Contributions to decision tree induction: bias/variance tradeoff and time series classification. Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of Liege, Belgium.

[13] Itakura, F. (1975). Minimum prediction residual principle applied to speech recognition. IEEE Trans. Acoustics, Speech, and Signal Proc., Vol. ASSP-23, pp. 52-72.

[14] Kadous, M.W. (1999). Learning comprehensible descriptions of multivariate time series. In Proceedings of the 16th International Machine Learning Conference. pp. 454-463

[15] Keogh, E. (2002). Exact indexing of dynamic time warping. In 28th International Conference on Very Large Data Bases. Hong Kong. pp. 406-417.

[16] Keogh, E., Chakrabarti, K., Pazzani, M., and Mehrotra, S. (2001). Locally adaptive dimensionality reduction for indexing large time series databases. In Proceeding so ACM SIGMOD International conference on Management of data, pp. 151-162.

[17] Keogh, E. and Kasetty, S. (2002). On the Need for Time Seires Data Mining Benchmarks: A Survey and Empirical Demonstration. In the 8th ACM SIGKDD, pp. 102-111.

[18] Kim, S.W., Park, S., & Chu, W.W. (2004). Efficient processing of similarity search under time warping in sequence databases: an index-based approach. Inf. Syst. 29(5): 405-420.

[19] Kornfield, E.M, Manmatha, R., and Allan, J. (2004). Text Alignment with Handwritten Documents. 1st International workshop on Document Image Analysis for Libraris (DIAL), pp. 195-209.

[20] Laaksonen, J., Hurri, J., and Oja, Erkki. (1998). Comparison of Adaptive Strategies for On-Line Character Recognition. In proceedings of ICANN'98, pp. 245-250.

[21] Park, S.,, Chu, W, Yoon, J., and Hsu, C (2000). Efficient searchs for similar subsequences of different lengths in sequence databases. In ICDE-00.

[22] Rabiner, L., Rosenberg, A. & Levinson, S. (1978). Considerations in dynamic time warping algorithms for

discrete word recognition. IEEE Trans. Acoustics Speech, and Signal Proc., Vol. ASSP-26, pp. 575-582.

[23] Ratanamahatana, C.A. & Keogh, E. (2004). Making Time-series Classification More Accurate Using Learned Constraints. In proceedings of SDM International conference, pp. 11-22.

[24] Rath, T. & Manmatha, R. (2003). Word image matching using dynamic time warping. CVPR, Vol. II, pp. 521-527.

[25] Roverso, D. (2000). Multivariate temporal classification by windowed wavelet decomposition and recurrent neural networks. In 3rd ANS International Topical Meeting on Nuclear Plant Instrumentation, Control and Human-Machine Interface, 2000.

[26] Sakoe, H. & chiba, S. (1978). Dynamic programming algorithm optimization fro spoken word recognition. IEEE Trans. Acoustics, Speech, and Signal Proc., Vol. ASSP-26. pp. 43-49.

[27] Shou, Y., Mamoulis, N., and Cheung, D.W. (2004). Efficient Warping of Segmented Time-series, HKU CSIS Tech report, TR-2004-01, March 2004.

[28] Wong, T.S.F & Wong, M.H. (2003). Efficient Subsequence Matching for Sequences Databases under Time Warping. IDEAS '03: 139-148.

[29] Yi, B.K., Jagadish, H. & Faloutsos, C. (1998). Efficient retrieval of similar time sequences under time warping. In ICDE '98, pp. 23-27.

[30] Zhu, Y. & Shasha, D. (2003). Warping Indexes with Envelope Transforms for Query by Humming. SIGMOD 2003. pp. 181-192.

10. APPENDIX A: A DESCRIPTION OF THE DATASETS USED

We have chosen seven classification datasets to be tested in our work. Note that we Z-normalized (mean = 0 and standard deviation = 1) all the datasets used in this work.

10.1 FACE DATASET

This is a face classification problem based on the head profiles. We took a number of photos (20-35) of four different individuals (1 female, 3 males) making different expressions on the face, e.g. talking, smiling, frowning, laughing, etc. We then convert each head profile, starting from the neck area into a "pseudo time series" by measuring the local angle of a trace of its perimeter, as shown in Figure 14. The dataset contains 112 instances in total with the length of each instance ranges from 107 to 240 data points.

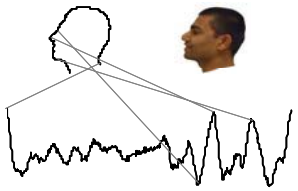


Figure 14. Starting from the neck area, the head profile is converted into a "pseudo time series"

10.2 GUN PROBLEM

This dataset comes from the video surveillance domain. The dataset has two classes, each containing 100 instances. All instances were created using one female actor and one male actor in a single session. The two classes are:

Gun-Draw: The actors have their hands by their sides. They draw a replicate gun from a hip-mounted holster, point it at a target for approximately one second, then return the gun to the holster, and their hands to their sides.

Point: The actors have their gun by their sides. They point with their index fingers to a target for approximately one second, and then return their hands to their sides.

For both classes, we tracked the centroid of the actor's right hands in both X- and Y-axes, which appear to be highly correlated; therefore, in this experiment, we only consider the X-axis for simplicity.

The overall motions of both classes are very similar. However, it is possible for human to visually classify the two classes with great accuracy, after noting that the actor must lift his/her hand above a holster, then reach down for the gun. This action creates a subtle distinction between the classes as shown in Figure 15.

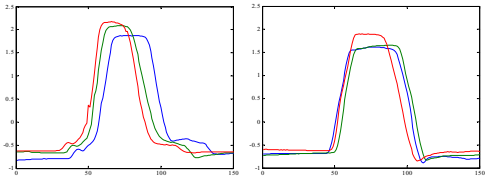


Figure 15. (left) some examples from the Gun-Draw data. (right) some examples from the Point data

The dataset contains 200 instances, 100 for each class. Since each actor was periodically signaled every 5 seconds before each repetition of Gun-Draw/Point, the video clip (captured at 30 frames per second) was easily segmented into 150 data points for each instance.

10.3 LEAF DATASET

This dataset contains a collection of six different species of leaf images, including 2 genera of plant, i.e. oak and maple (all original images can be found at [http://web.engr.oregonstate.edu/~tgd/leaves/dataset/herbari

um]). The dataset comprises four different species of Maple and two species of Oak, with 442 instances in total. We convert each leaf image into "pseudo time series" using similar method as the Face Dataset. Figure 11 shows an example of a Glabrum maple image converted into a 'time series'. The length of each time series ranges from 22 to 475 data points.

10.4 SYNTHETIC CONTROL CHART

This six-class dataset was retrieved from the UCR Time Series Data Mining Archive [http://www.cs.ucr.edu/~eamonn/TSDMA/]. It contains 600 instances in total with 100 instances in each class. Each instance has the same length of 60 data points.

10.5 TRACE DATASET

This dataset is a subset of the Transient Classification Benchmark first introduced by Davide Roverso [25]. This is a synthetic dataset designed to simulate instrumentation failures in a nuclear power plant. The full dataset consists of 16 classes, 50 instances in each class. Each instance has 4 features.

For simplicity, we only use the second feature of class 2 and 6, and the third feature of class 3 and 7 for our experiment. Our dataset contains 200 instances, 50 for each class. The length of each instance ranges between 279 and 386 data points.

10.6 TWO-PATTERN DATASET

This four-class dataset contains 5,000 instances. Each instance has the same length of 128 data points. The dataset was introduced in [12]. Each class is characterized by the presence of two patterns in a definite order, down-down, up-down, down-up, and up-up.

10.7 WORDSPOTTING DATASET

This is a subset of the WordSpotting Project dataset created by Rath and Manmatha [24]

In the full dataset, there are 2,381 words with four features that represent each word image's profiles or the background/ink transitions.

For simplicity, we pick the "Projection Profile" (feature 1) of the four most common words, "the", "to", "be", and "that", to be used in our experiment. "the" has 109 instances; "to" has 91 instances; "be" has 38 instances, and "that" has 34 instances. Once combined, we obtain a dataset of 272 instances, with the length of each instance ranges from 41 to 192 data points.

FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space

Stan Salvador and Philip Chan
Dept. of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901

{ssalvado, pkc}@cs.fit.edu

ABSTRACT

The dynamic time warping (DTW) algorithm is able to find the optimal alignment between two time series. It is often used to determine time series similarity, classification, and to find corresponding regions between two time series. DTW has a quadratic time and space complexity that limits its use to only small time series data sets. In this paper we introduce FastDTW, an approximation of DTW that has a linear time and space complexity. FastDTW uses a multilevel approach that recursively projects a solution from a coarse resolution and refines the projected solution. We prove the linear time and space complexity of FastDTW both theoretically and empirically. We also analyze the accuracy of FastDTW compared to two other existing approximate DTW algorithms: Sakoe-Chuba Bands and Data Abstraction. Our results show a large improvement in accuracy over the existing methods.

Keywords

dynamic time warping, time series

1. INTRODUCTION

Motivation. Dynamic time warping (DTW) is a technique that finds the optimal alignment between two time series if one time series may be "warped" non-linearly by stretching or shrinking it along its time axis. This warping between two time series can then be used to find corresponding regions between the two time series or to determine the similarity between the two time series. Dynamic time warping is often used in speech recognition to determine if two waveforms represent the same spoken phrase. In a speech waveform, the duration of each spoken sound and the interval between sounds are permitted to vary, but the overall speech waveforms must be similar. In addition to speech recognition, dynamic time warping has also been found useful in many other disciplines [8], including data mining, gesture recognition, robotics, manufacturing, and medicine. Dynamic time warping is commonly used in data mining as a distance measure between time series. An example of how one time series is "warped" to another is shown in Figure 1.

In Figure 1, each vertical line connects a point in one time series to its correspondingly similar point in the other time series. The lines actually have similar values on the *y*-axis but have been separated so the vertical lines between them can be viewed more easily. If both of the time series in Figure 1 were identical, all of the lines would be straight vertical lines because no warping would be necessary to 'line up' the two time series. The warp path distance is a measure of the difference between the two time series after they have been warped together, which is measured by the sum of the distances between each pair of points connected by the vertical lines in Figure 1. Thus, two time series that are identical except for localized stretching of the time axis will have DTW distances of zero.

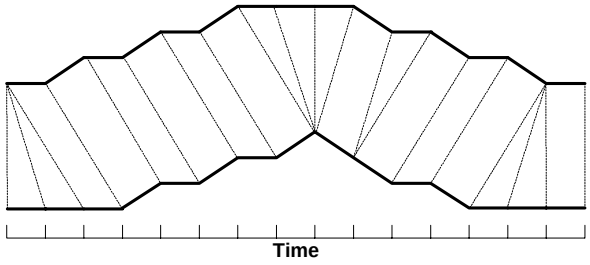


Figure 1. A warping between two time series.

Despite the effectiveness of the dynamic time warping algorithm, it has an $O(N^2)$ time and space complexity that limits its usefulness to small time series containing no more than a few thousand data points. More details of the dynamic time warping algorithm are contained in Section 2.1.

Problem. We desire to develop a dynamic time warping algorithm that is linear in both time and space complexity and can find a warp path between two time series that is nearly optimal.

Approach. In this paper we introduce the FastDTW algorithm, which is able to find an accurate approximation of the optimal warp path between two time series. The FastDTW algorithm avoids the brute-force dynamic programming approach of the standard DTW algorithm by using a multilevel approach. The time series are initially sampled down to a very low resolution. A warp path is found for the lowest resolution and "projected" onto an incrementally higher resolution time series. The projected warp path is refined and projected again to yet a higher resolution. The process of refining and projecting is continued until a warp path is found for the full resolution time series.

Contributions. Our main contribution is the introduction of the FastDTW algorithm, which is an accurate approximation of DTW that runs in linear time and space. We prove the $O(N)$ time and space complexity both theoretically and empirically. We also empirically demonstrate that FastDTW produces an accurate minimum-distance warp path between two time series than is nearly optimal (standard DTW is optimal, but has a quadratic time and space complexity). In addition to the FastDTW algorithm, we evaluate other existing approximate DTW algorithms, and compare their accuracy on a large and diverse group of time series data sets.

Organization. The next section describes the standard dynamic time warping algorithm and existing approaches to speed it up. Section 3 provides a detailed explanation of our FastDTW algorithm. Section 4 discusses experimental evaluations of the FastDTW algorithm based on accuracy, and time/space complexity, and Section 5 summarizes our study.

2. RELATED WORK

2.1 Dynamic Time Warping (DTW)

A distance measurement between time series is needed to determine similarity between time series and for time series classification. Euclidean distance is an efficient distance measurement that can be used. The Euclidian distance between two time series is simply the sum of the squared distances from each *n*th point in one time series to the *n*th point in the other. The main disadvantage of using Euclidean distance for time series data is that its results are very unintuitive. If two time series are identical, but one is shifted slightly along the time axis, then Euclidean distance may consider them to be very different from each other. Dynamic time warping (DTW) was introduced [11] to overcome this limitation and give intuitive distance measurements between time series by ignoring both global and local shifts in the time dimension.

Problem Formulation. The dynamic time warping problem is stated as follows: Given two time series X, and Y, of lengths $|X|$ and $|Y|$,

$$X = x_1, x_2, \ldots, x_i, \ldots, x_{|X|}$$

$$Y = y_1, y_2, \ldots, y_j, \ldots, y_{|Y|}$$

construct a warp path W

$$W = w_1, w_2, \ldots, w_K \qquad \max(|X|, |Y|) \le K < |X| + |Y|$$

where K is the length of the warp path and the k^{th} element of the warp path is

$$w_k = (i, j)$$

where i is an index from time series X, and j is an index from time series Y. The warp path must start at the beginning of each time series at $w_1 = (1, 1)$ and finish at the end of both time series at $w_K = (|X|, |Y|)$. This ensures that every index of both time series is used in the warp path. There is also a constraint on the warp path that forces i and j to be monotonically increasing in the warp path, which is why the lines representing the warp path in Figure 1 do not overlap. Every index of each time series must be used. Stated more formally:

$$w_k = (i, j), w_{k+1} = (i', j') \qquad i \le i' \le i+1, \quad j \le j' \le j+1$$

The optimal warp path is the warp path is the minimum-distance warp path, where the distance of a warp path W is

$$Dist(W) = \sum_{k=1}^{k=K} Dist(w_{ki}, w_{kj})$$

$Dist(W)$ is the distance (typically Euclidean distance) of warp path W, and $Dist(w_{ki}, w_{kj})$ is the distance between the two data point indexes (one from X and one from Y) in the k^{th} element of the warp path.

DTW Algorithm. A dynamic programming approach is used to find this minimum-distance warp path. Instead of attempting to solve the entire problem all at once, solutions to sub-problems (portions of the time series) are found, and used to repeatedly find solutions to a slightly larger problem until the solution is found for the entire time series. A two-dimensional $|X|$ by $|Y|$ cost matrix D, is constructed where the value at $D(i, j)$ is the minimum-distance warp path that can be constructed from the two time series $X'=x_1,...,x_i$ and $Y'=y_1,...,y_j$. The value at $D(|X|, |Y|)$ will contain the minimum-distance warp path between time series X and Y. Both axes of D represent time. The x-axis is the time of time series X, and the y-axis is the time of time series Y. Figure 2 D shows an example of a cost matrix and a minimum-distance warp path traced through it from $D(1, 1)$ to $D(|X|, |Y|)$.

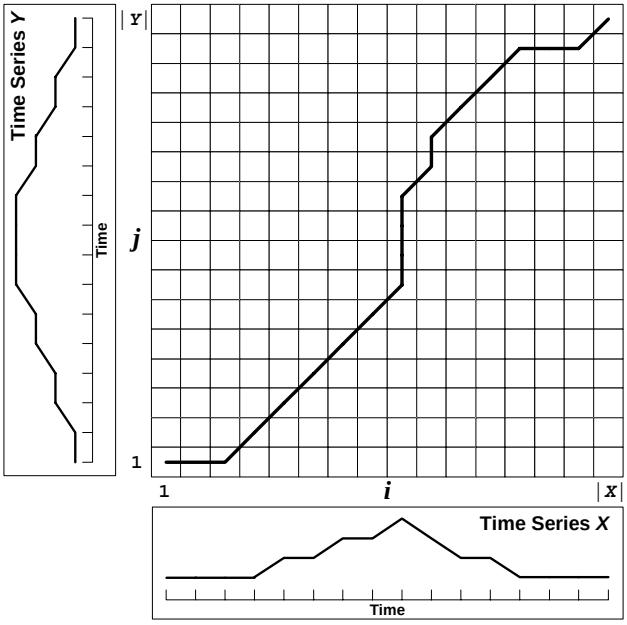


Figure 2. A cost matrix with the minimum-distance warp path traced through it.

The cost matrix and warp path in Figure 2 are for the same two time series shown in Figure 1. The warp path is $W = \{(1,1), (2,1), (3,1), (4,2), (5,3), (6,4), (7,5), (8,6), (9,7), (9,8), (9,9), (9,10), (10,11), (10,12), (11,13), (12,14), (13,15), (14,15), (15,15), (16,16)\}$. If the warp path passes through a cell $D(i, j)$ in the cost matrix, it means that the i^{th} point in time series X is warped to the j^{th} point in time series Y. Notice that where there are vertical sections of the warp path, a single point in time series X is warped to multiple points in time series Y, and the opposite is also true where the warp path is a horizontal line. Since a single point may map to multiple points in the other time series, the time series do not need to be of equal length. If X and Y were identical time series, the warp path through the matrix would be a straight diagonal line.

To find the minimum-distance warp path, every cell of the cost matrix must be filled. The rationale behind using a dynamic programming approach to this problem is that since the value at $D(i, j)$ is the minimum warp distance of two time series of lengths i and j, if the minimum warp distances are already known for all

slightly smaller portions of that time series that are a single data point away from lengths *i* and *j*, then the value at $D(i, j)$ is the minimum distance of all possible warp paths for time series that are one data point smaller than *i* and *j*, plus the distance between the two points x_i and y_j. Since the warp past must either be incremented by one or stay the same along the *i* and *j* axes, the distances of the optimal warp paths one data point smaller than lengths *i* and *j* are contained in the matrix at $D(i\text{-}1, j)$, $D(i, j\text{-}1)$, and $D(i\text{-}1, j\text{-}1)$. So the value of a cell in the cost matrix is:

$$D(i, j) = Dist(i, j) + \min[D(i-1, j), D(i, j-1), D(i-1, j-1)]$$

The warp path to $D(i, j)$ must pass through one of those three grid cells, and since the minimum possible warp path distance is already known for them, all that is needed is to simply add the distance of the current two points to the smallest one. Since this equation determines the value of a cell in the cost matrix by using the values in other cells, the order that they are evaluated in is very important. The cost matrix is filled one column at a time from the bottom up, from left to right as depicted in Figure 3.

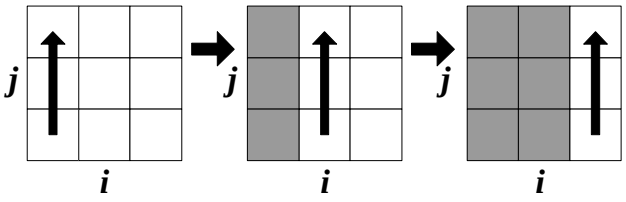


Figure 3. The order that the cost matrix is filled.

After the entire matrix is filled, a warp path must be found from $D(1, 1)$ to $D(|X|, |Y|)$. The warp path is actually calculated in reverse order starting at $D(|X|, |Y|)$. A greedy search is performed that evaluates cells to the left, down, and diagonally to the bottom-left. Whichever of these three adjacent cells has the smallest value is added to the beginning of the warp path found so far, and the search continues from that cell. The search stops when $D(1, 1)$ is reached.

Complexity of DTW. Time and Space complexity of the DTW is easy to determine. Each cell in the $|X|$ by $|Y|$ cost matrix is filled exactly once, and each cell is filled in constant time. This yields both a time and space complexity of $|X|$ by $|Y|$, which is $O(N^2)$ if $N=|X|=|Y|$. The quadratic space complexity is particularly prohibitive because memory requirements are in the *terabyte* range for time series containing only 177,000 measurements. A linear space-complexity implementation of the DTW algorithm is possible by only keeping the current and previous columns in memory as the cost matrix is filled from left to right (see Figure 3). By only retaining two columns at any one time, the optimal warp distance between the two time series can be determined. However it is not possible to reconstruct the warp path between these two time series because the information required to calculate the warp path is thrown away with the discarded columns. This is not a problem if only the distance between two time series is required, but applications that find corresponding regions between time series [14] or merge time series together [1][3] require the warp path to be found.

2.2 Speeding up Dynamic Time Warping

The quadratic time and space complexity of DTW creates the need for methods to speed up dynamic time warping. The methods used make DTW faster fall into three categories:

1) *Constraints* – Limit the number of cells that are evaluated in the cost matrix.
2) *Data Abstraction* – Perform DTW on a reduced representation of the data.
3) *Indexing* – Use lower bounding functions to reduce the number of times DTW must be run during time series classification or clustering.

Constraints are widely used to speed up DTW. Two of the most commonly used constraints are the Sakoe-Chuba Band [13] and the Itakura Parallelogram [4], which are shown in Figure 4.

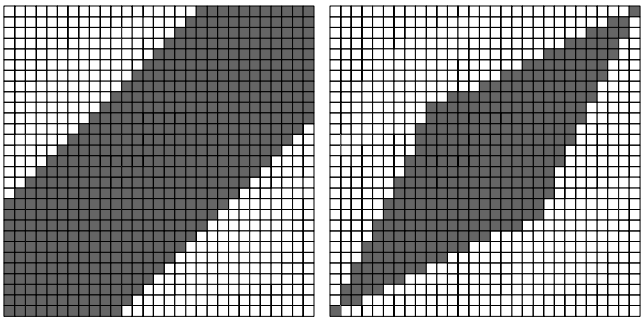


Figure 4. Two constraints: Sakoe-Chuba Band (left) and an Itakura Parallelogram (right), both have a width of 5.

The shaded areas in Figure 4 are the cells of the cost matrix that are filled in by the DTW algorithm for each constraint. The width of each shaded area, or window, is specified by a parameter. When constraints are used, the DTW algorithm finds the optimal warp path through the constraint window. However, the globally optimal warp path will not be found if it is not entirely inside the window. Using constraints speeds up DTW by a constant factor, but the DTW algorithm is still $O(N^2)$ if the size of the input window is a function of the length of the input time series. Constraints work well in domains where the optimal warp path is expected to be close to a linear warp and passes through the cost matrix diagonally in a relatively straight line. Constraints work poorly if time series are of events that start and stop at radically different times because the warp path can stray very far from a linear warp and nearly the entire cost matrix must be evaluated to find the optimal warp path.

Data abstraction speeds up the DTW algorithm by running DTW on a reduced representation of the data [2][9]. The left side of Figure 5 shows a full-resolution cost matrix for which a minimum-distance warp path must be found. Rather than running the DTW algorithm on the full resolution (1/1) cost matrix, the time series are reduced in size to make the number of cells in the cost matrix more manageable. A warp path is found for the lower-resolution time series and is mapped back to the full resolution cost matrix.

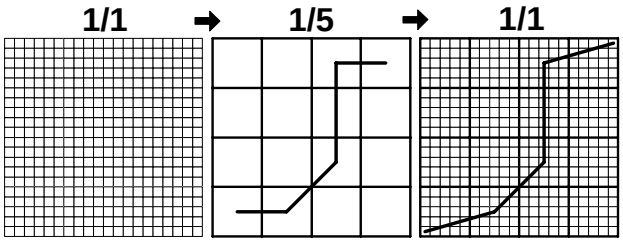


Figure 5. Speeding up DTW by data abstraction.

The result is that DTW is sped up by a large constant factor, but the algorithm still runs in $O(N^2)$ time and space. Obviously, the warp distance that is calculated between the two time series becomes increasingly inaccurate as the level of abstraction increases. Projecting the lower resolution warp path to the full resolution usually creates a warp path that is far from optimal because even *IF* the optimal warp path actually passes through the low-resolution cell, projecting the warp path to the higher resolution ignores local variations in the warp path that can be very significant.

Indexing uses lower-bounding functions to prune out the number of times DTW needs to be run for certain tasks such as clustering a set of time series or finding the time series that is most similar to a given time series [6][10]. Indexing significantly speeds up many DTW applications by reducing the number of times DTW is run, but does not speed up the actual DTW algorithm.

Our FastDTW algorithm uses ideas from both the constraints and data abstraction categories. Using a combination of both overcomes many limitations of using either method individually, and yields an algorithm that is $O(N)$ in both time and space.

3. APPROACH

The multilevel approach that FastDTW uses is inspired by the multilevel approach used for graph bisection [5]. Graph bisection is the task of splitting a graph into roughly equal portions, such that the sum of the edges that would be broken is as small as possible. Efficient and accurate algorithms exist for small graphs, but for large graphs, the solutions found are typically far from optimal. A multilevel approach can be used to find the optimal solution for a small graph, and then repeatedly expand the graph and "fix" the pre-existing solution for the slightly larger problem. A multilevel approach works well if a large problem is difficult to solve all at once, but partial solutions can effectively be refined at different levels of resolution. The dynamic time warping problem can also be solved with a multilevel approach. Our FastDTW algorithm uses the multilevel approach and is able to find an accurate warp path in linear time and space.

3.1 FastDTW Algorithm

The FastDTW algorithm uses a multilevel approach with three key operations:

1) ***Coarsening*** – Shrink a time series into a smaller time series that represents the same curve as accurately as possible with fewer data points.

2) ***Projection*** – Find a minimum-distance warp path at a lower resolution, and use that warp path as an initial guess for a higher resolution's minimum-distance warp path.

3) ***Refinement*** – Refine the warp path projected from a lower resolution through local adjustments of the warp path.

Coarsening reduces the size (or resolution) of a time series by averaging adjacent pairs of points. The resulting time series is a factor of two smaller than the original time series. Coarsening is run several times to produce many different resolutions of the time series. *Projection* takes a warp path calculated at a lower resolution and determines what cells in the next higher resolution time series the warp path passes through. Since the resolution is increasing by a factor of two, a single point in the low-resolution warp path will map to at least four points at the higher resolution (possibly >4 if $|X| \neq |Y|$). This projected path is then used as a heuristic during solution refinement to find a warp path at the higher resolution. *Refinement* finds the optimal warp path *in the neighborhood* of the projected path, where the size of the neighborhood is controlled by the *radius* parameter.

Standard dynamic time warping (DTW) is an $O(N^2)$ algorithm because every cell in the cost matrix must be filled to ensure an optimal answer is found, and the size of the matrix grows quadratically with the size of the time series. In the multilevel approach, the cost matrix is only filled in the neighborhood of the path projected from the previous resolution. Since the length of the warp path grows *linearly* with the size of the input time series, the multilevel approach is an $O(N)$ algorithm.

The FastDTW algorithm first uses coarsening to create all of the resolutions that will be evaluated. Figure 6 shows four resolutions that are created when running the FastDTW algorithm on the time series that were previously used in Figures 1 and 2. The standard DTW algorithm is run to find the optimal warp path for the lowest resolution time series. This lowest resolution warp path is shown in the left of Figure 6. After the warp path is found for the lowest resolution, it is projected to the next higher resolution. In Figure 6, the projection of the warp path from a resolution of 1/8 is shown as the heavily shaded cells at 1/4 resolution.

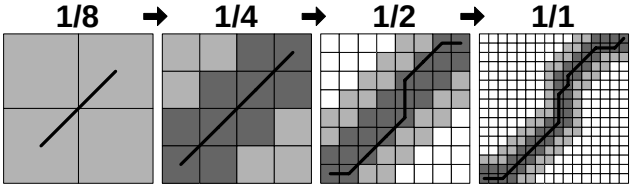


Figure 6. The four different resolutions evaluated during a complete run of the FastDTW algorithm.

To refine the projected path, a constrained DTW algorithm is run with the very specific constraint that only cells in the projected warp path are evaluated. This will find the optimal warp path *through the area of the warp path that was projected from the lower resolution*. However, the entire optimal warp path may not be contained within projected path. To increase the chances of finding the optimal solution, there is a *radius* parameter that controls the *additional* number of cells on each side of the projected path that will also be evaluated when refining the warp path. In Figure 6, the *radius* parameter is set to 1. The cells included during warp path refinement due to the *radius* are lightly

shaded. Once the warp path is refined at the 1/4 resolution, that warp path is projected to the 1/2 resolution, expanded by a *radius* of 1, and refined again. Finally, the warp path is projected to the full resolution (1/1) matrix in Figure 6. The projection is expanded by the *radius* and refined one last time. This refined warp path is the output of the algorithm.

Notice that the warp path found by the FastDTW algorithm in Figure 6 is the optimal warp path that was found by the standard DTW in Figure 2. However, FastDTW only evaluated the shaded cells, while DTW evaluates all of the cells in the cost matrix. FastDTW evaluated 4+16+44+100=164 cells at all resolutions, while DTW evaluates all 235 (16^2) cells. This increase in efficiency is not very significant for his small problem, especially considering the overhead of creating all four resolutions. However, the number of cells that FastDTW evaluates scales linearly with the length of the time series, while DTW always evaluates N^2 cells (if both time series are of length N). FastDTW scales linearly because the width of the path through the matrix that is being evaluated is constant at all resolutions.

The example in Figure 6 finds the optimal warp path, but the FastDTW algorithm is not guaranteed to always find a warp path that is optimal. However, the path found is usually very close to optimal. The larger the value of the *radius* parameter, the more accurate the warp path will be. If the *radius* parameter is set to be as large as one of the input time series, then FastDTW generalizes to the DTW algorithm (optimal but $O(N^2)$). The accuracy of FastDTW using different settings for the *radius* parameter will be demonstrated in Section 4.

The pseudocode for the FastDTW algorithm is shown Figure 7. The input to the algorithm is two time series, and the *radius* parameter. The output of FastDTW is a warp path and the distance between the two time series along that warp path. Line 2 determines the minimum length of a time series at the lowest resolution. This size is dependent on the *radius* parameter and determines the smallest possible resolution size for which decreasing the resolution further would be pointless because full dynamic time warping would need to be calculated at more than one resolution.

FastDTW has a straightforward recursive implementation. The base case is when one of the input time series has a length less than *minTSsize*. For the base case, the algorithm simply returns the result of the standard DTW algorithm. The recursive case has three main steps. First, two new lower-resolution time series are created that have half as many points as the input time series (***coarsening***). This is performed by lines 17-18 in Figure 7. Next, a low resolution path is found for the coarsened time series (lines 20-21) and ***projected*** to a higher resolution (lines 23-25). This projected path is also expanded by *radius* cells to create a search window that will be passed to a constrained version of the DTW algorithm that only evaluates the cells in the search window (line 27). The constrained DTW algorithm ***refines*** the warp path that was projected form the lower resolution. The result of this refinement is then returned.

Function FastDTW()

Input: *X* – a TimeSeries of length |*X*|
Y – a TimeSeries of length |*Y*|
radius – distance to search outside of the projected warp path from the previous resolution when refining the warp path

Output: 1) A min. distance warp path between *X* and *Y*
2) The warped path distance between *X* and *Y*

```
 1| // The min size of the coarsest resolution.
 2| Integer minTSsize = radius+2
 3|
 4| IF (|X|≤minTSsize OR |Y|≤minTSsize)
 5| {
 6|   // Base Case: for a very small time series run
 7|   //   the full DTW algorithm.
 8|   RETURN DTW(X, Y)
 9| }
10| ELSE
11| {
12|   // Recursive Case: Project the warp path from
13|   //   a coarser resolution onto the current
14|   //   current resolution. Run DTW only along
15|   //   the projected path (and also 'radius' cells
16|   //   from the projected path).
17|   TimeSeries shrunkX = X.reduceByHalf()
18|   TimeSeries shrunkY = Y.reduceByHalf()
19|
20|   WarpPath lowResPath =
21|       FastDTW(shrunkX,shrunkY, radius)
22|
23|   SearchWindow window =
24|       ExpandedResWindow(lowResPath, X, Y,
25|                          radius)
26|
27|   RETURN DTW(X, Y, window)
28| }
```

Figure 7. The FastDTW algorithm.

The execution of the FastDTW algorithm repeatedly runs lines 17-18 in recursive calls to lower resolutions are made by line 21. This creates multiple resolutions until the base case is reached (line 8). The base case is executed only a single time, and afterwards lines 23-27 are executed for each recursive call (or resolution) on the stack.

Next, we will provide a theoretical analysis of FastDTW based on time and space complexity.

Time Complexity of FastDTW. To simplify the calculations we will assume that the two full-resolution time series *X* and *Y* are both of length *N*. All analysis will be performed on worst-case behavior.

The number of cells in the cost matrix that are filled by FastDTW in a single resolution is equal the number of cells in the projected warp path and any other cells within *radius* (denoted as *r* in the rest of this analysis to save space) cells away from the projected path. The worst case, a straight diagonal projected warp path is depicted in Figure 8.

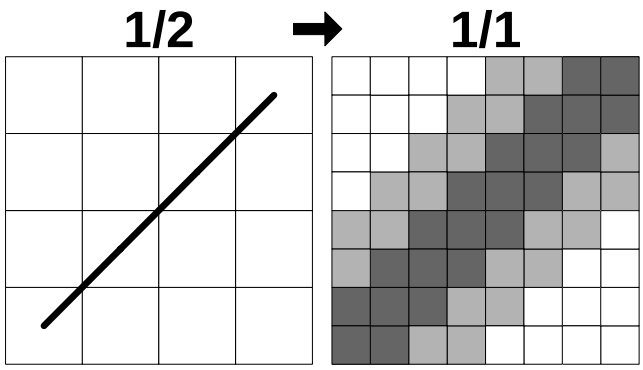


Figure 8. Maximum (worst-case) number of cells evaluated for a *radius* of 1.

The lightly shaded cells in Figure 8 are the 2*Nr* cells on each side of the projected path (heavily shaded cells), which itself has 3*N* cells. The projected path therefore has the following maximum number of cells at a resolution with two time series containing *N* points:

$$3N + 2(2Nr) = N(4r + 3) \quad [1]$$

The length of the time series at each resolution (*res*) follows the sequence (*N* points are contained in the original time series):

$$\left\{ \frac{N}{2^{res}} \right\}_{res=0}^{res=\infty} = N, \frac{N}{2}, \frac{N}{2^2}, \frac{N}{2^3}, \frac{N}{2^4}, \cdots \quad [2]$$

Therefore, the number of cells evaluated at all resolutions is (combine Equations 1 and 2)

$$\sum_{res=0}^{\infty} \frac{N}{2^{res}}(4r+3) = N(4r+3) + \frac{N}{2}(4r+3) + \frac{N}{2^2}(4r+3) + \cdots \quad [3]$$

The series in Equation 3 is very similar to the series

$$\sum_{res=0}^{\infty} \frac{1}{2^{res}} = 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \cdots = 2 \quad [4]$$

Multiplying Equation 4 by Equation 1 yields

$$N(4r+3) + \frac{N}{2}(4r+3) + \frac{N}{2^2}(4r+3) + \cdots = 2N(4r+3) \quad [5]$$

Since the sequence in Equation 5 is identical to the sequence in Equation 3, the number of cells evaluated at all resolutions is

$$\textbf{\textit{Total number of cells filled}} = 2N(4r+3) \quad [6]$$

In addition to the number of cells calculated there is also time complexity for creating the coarser resolutions and determining the warp path by tracing through the matrix.

The time complexity needed to create the resolutions is proportional to the number of points in all of the resolutions, which is the series in Equation 2. The solution of Equation 2 is obtained by multiplying Equation 4 by *N*, which yields 2*N*. Since multiple resolutions of both time series must be created, 2*N* is multiplied by two to get the final time complexity.

$$\textbf{\textit{Time to create all resolutions}} = 4N \quad [7]$$

The time complexity needed to trace the warp path back through a matrix is measured by the length of the warp path. A resolution containing *N* points has a length of 2*N* in the worst case (*N* is the best case for a diagonal line). Multiplying Equation 4 by 2*N* gives the worst-case length of all warp paths added together from every resolution:

$$\textbf{\textit{Time to trace warp paths}} = 4N \quad [8]$$

Adding Equations 6, 7, and 8 gives the total worst-cast time complexity of FastDTW

$$\textbf{\textit{FastDTW time complexity}} = N(8r+14) \quad [9]$$

which is O(*N*) if *r* (*radius*) is a small constant value.

Space Complexity of FastDTW. The space complexity of FastDTW consists of the space required to store the resolutions (other than the full-resolution input time series), the maximum amount of cells that are used at any one time in a cost matrix, and the size of the warp path stored In memory. The space complexity of storing all extra resolutions other than the full resolution for one input time series is Equation 2 without the first them, which is 2*N*-*N*=*N*. For both input time series the space complexity is

$$\textbf{\textit{Space of resolutions (other that full resolution)}} = 2N \quad [10]$$

The space complexity of the cost matrix is the maximum size cost matrix that is created for the full resolution matrix. The number of cells in the matrix is Equation 1

$$\textbf{\textit{Space of cost matrix}} = N(4r+3) \quad [11]$$

The space complexity of storing the warp path is equal to the longest warp path that can exist at full resolution. If the warp path traces the perimeter of the cost matrix, then the length of that path will be

$$\textbf{\textit{Space complexity of storing the warp path}} = 2N \quad [12]$$

And adding Equations 10, 11, and 12 gives the total worst-case space complexity of

$$\textbf{\textit{FastDTW space complexity}} = N(4r+7) \quad [13]$$

which is also O(*N*) if *r* (*radius*) is a small (<*N*) constant value.

4. EMPIRICAL EVALUATION

The goal of this evaluation is demonstrate the efficiency and accuracy of the FastDTW algorithm on a wide range of time series data sets. To ensure reproducibility, all datasets and algorithms used in this evaluation can be found online at "http://cs.fit.edu/~pkc/FastDTW/". This evaluation will first demonstrate the accuracy of the FastDTW algorithm and will then empirically verify its linear time complexity.

4.1 Accuracy of FastDTW

4.1.1 Procedures and Criteria

The accuracy of an approximate DTW algorithm can be measured by determining how much the approximate warp path distance differs from the optimal warp path distance. The error of an

approximate DTW algorithm, such as our FastDTW algorithm, is calculated by the following equation:

$$\textbf{\textit{Error of a warp path}} = \frac{approxDist - optimalDist}{optimalDist} \times 100 \quad [14]$$

If the DTW algorithm finds a warp path with a distance equal to the optimal warp path distance, then there is zero error. The optimal warp path distance can be found by running the standard DTW algorithm. The error of a warp path will always be ≥0% (because *optimalDist* is never larger than *approxDist*) and can exceed 100% if the distance of the approximate warp path is more than double the optimal distance.

The FastDTW algorithm is evaluated against two other existing approximate DTW algorithms: Sakoe-Chuba bands and data abstraction. Sakoe-Chuba bands (see left side of Figure 4) constrain the DTW algorithm to only evaluate a specified *radius* away from a linear warp within the cost matrix. Itakura Parallelograms (see right side of Figure 4) are not evaluated because, for a given *radius*, a band will always find a warp path equal to or better than that of the parallelogram. This is because the parallelogram constraint is a subset of the band constraint. The data abstraction DTW algorithm used in this evaluation first samples the data, and then runs the standard DTW algorithm to find a warp path on the sampled data. This warp path is then projected to the full resolution as previously shown in Figure 5.

The *radius* parameter performs a similar function for all three algorithms. It expands the region of the cost matrix searched from an initial "guess". For bands, the initial guess is a linear warp. For data abstraction, it is the projected warp path from the sampled data, and for FastDTW it is the projected warp path from the previous resolution. Each algorithm will be run with multiple *radius* parameters on a wide range of data sets.

All three algorithms (FastDTW, bands, and data abstraction) are only being evaluated based on accuracy in this section. However, care has been taken to ensure that the time each algorithm requires to execute is similar for the same *radius*. The data abstraction algorithm is made O(*N*) by sampling the data down to $\sqrt{N}$ points before performing quadratic time warping ($O(\sqrt{N}^2) = O(N)$). All three algorithms evaluate roughly the same number of cells in the cost matrix for any particular *radius*. FastDTW has some overhead for evaluating previous resolutions, and data abstraction has overhead for running standard DTW on the sampled time series. However, all three algorithms are linear with respect to the length of the input time series, and the number of cells evaluated for a given *radius* does not differ by more than a power of two of for any pair of algorithms.

The time series data sets used to evaluate the accuracy of the FastDTW algorithm include very similar data sets that are from the same domain, and dissimilar data sets that are from different domains. Both types of data are used to show that FastDTW works well on a wide range of data, regardless of the similarity or characteristics of the time series. Dynamic time warping is most frequently used to compare the similarity between time series, so it is likely that the majority of time series that are compared are similar and from the same domain. However, very dissimilar time series are also evaluated to ensure that the approximate FastDTW algorithm works well when warping two time series that do not share common features. The accuracy of each DTW algorithm is measured on three groups of data:

1) *Random* – 990 time warps between 45 time series from different domains (eeg, random walk, earthquake, speech, tide, etc.). The average length is 1128 points.

2) *Trace* - 10,900 time warps between 200 time series data sets. The Gun domain contains 4 classes that simulate instrumentation failure in a nuclear power plant. All time series have a length of 275 points.

3) *Gun* – 10,900 time warps between 200 time series data sets. The Gun domain contains 2 classes, with 100 time series of a gun being drawn from a holster and 100 time series of a gun being pointed. All time series have a length of 151 points.

All data sets used in this evaluation were obtained from the UCR Time Series Data Mining Archive and are publicly available [7]. Each algorithm and group of data is also run multiple times with the following settings for the *radius* parameter: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, and 30. For a given algorithm, group of data, and *radius*, the average error of all possible warp paths between time series in the group are recorded.

4.1.2 Results and Analysis

The FastDTW algorithm is very accurate for all three groups of data that it was tested on. FastDTW has an error of only 19.2% to 0.0%, depending on the value of the *radius* parameter. For all algorithms, the error decreases as the *radius* parameter increases. However, FastDTW converges to 0% error much faster than the other two algorithms. A summary of the results for several *radius* settings is contained in Table 1.

Table 1. Average error of three the algorithms at selected *radius* values (errors of the 3 groups of data are averaged).

	radius				
	0	*1*	*10*	*20*	*30*
FastDTW	19.2%	8.6%	1.5%	0.8%	0.6%
Abstraction	983.3%	547.9%	6.5%	2.8%	1.8%
Band	2749.2%	2385.7%	794.1%	136.8%	9.3%

Table 1 shows the average error for all three algorithms over all three test cases, when run with the *radius* set to 0, 1, 10, 20, and 30. FastDTW has a small amount of error for all *radius* settings, and begins to approach 0% error when *radius* is set at or above 10. Data abstraction is inaccurate for small *radius* values, but begins to be reasonably accurate when run with larger *radius* settings. The band algorithm is very inaccurate for all *radius* settings except for 30.

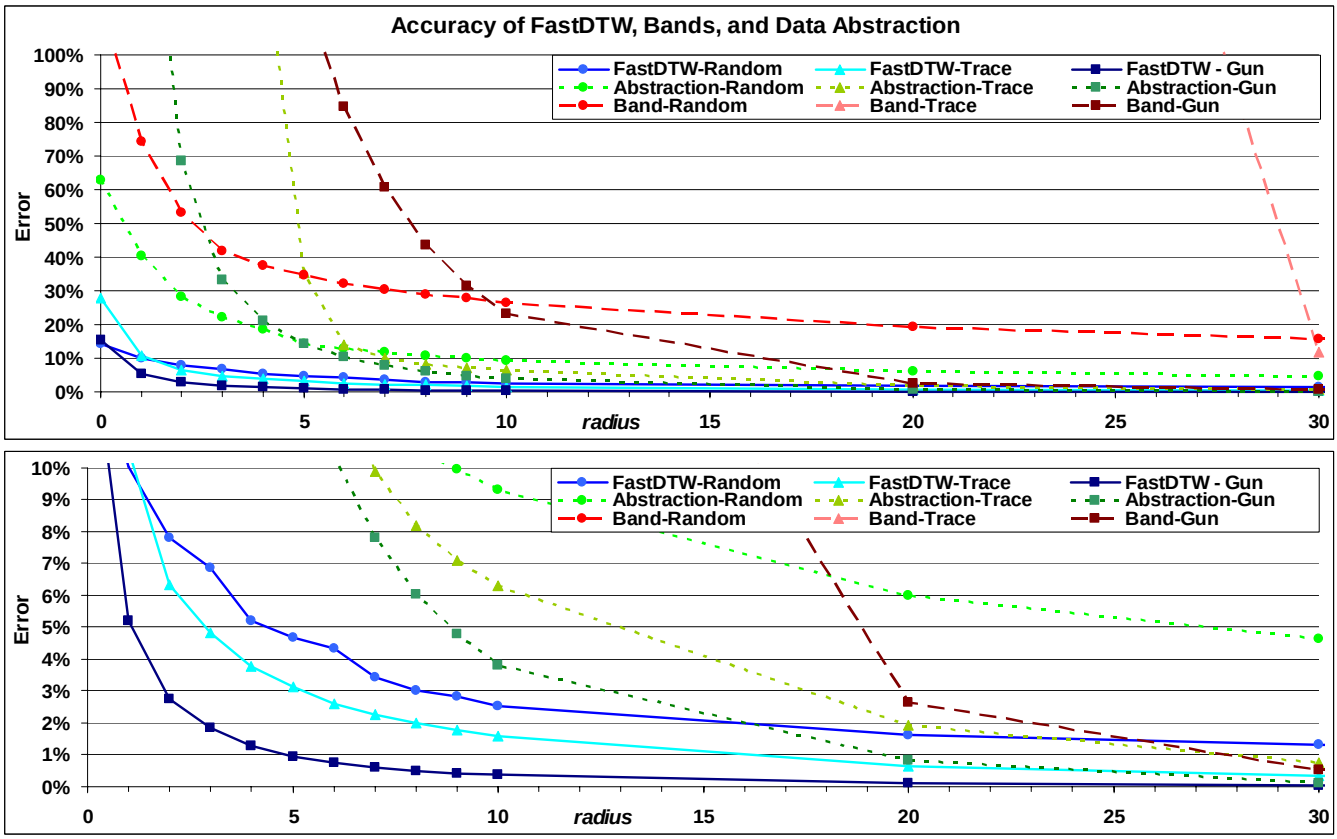


Figure 9. Accuracy of FastDTW compared to Bands and Data Abstraction. The top figure's *y*-axis is 0%-100% and the bottom figure's *y*-axis is 0%-10%.

Data abstraction is inaccurate (500-1000% error) for small *radius* settings because it blindly projects the warp path from a sampled time series onto a full resolution cost matrix. This projection may be "in the neighborhood" of a near-optimal warp path, but it fails to take into consideration any local variation in the warp path that is obscured by sampling. Local variations in the warp path can have a huge impact on the accuracy of a warp path. Increasing the *radius* setting (which is not part of the original data abstraction algorithm, it is introduced in this paper), can make it rather accurate because this begins to adjust the warp path to cover local variations. However, the accuracy is still worse than FastDTW for a given *radius* because FastDTW projects the "neighborhood" of the near-optimal warp path from the previous resolution in several small steps rather than a single large step.

Bands can only have good results if a near-optimal warp path is entirely contained within *radius* cells from a linear warp. When bands are used with a *radius* of 0, and the two time series are of equal length, it generalizes to Euclidean distance...which is a notoriously inaccurate similarity measure for time series [15]. A slight misalignment between the two time series being warped can cause a very large amount of error in the warp path.

The accuracy of each algorithm on the different groups of data is displayed in Figure 9.

In Figure 9, the *x*-axis is the radius parameter used, and the *y*-axis is the error of the tested algorithm. Each of the 9 lines is a combination between the three algorithms and the three groups of data sets. The FastDTW algorithm curves are solid lines, data abstraction curves are dotted lines, and band curves are dashed lines. The three groups of data can be identified by the shape of the markers on the curves. Round markers are used on curves using Random data, triangle markers are for the Trace data, and square markers are for the Gun data.

The three solid lines at the bottom of Figure 9 are the error curves for FastDTW on all three groups of data. The error is small for all three lines, meaning that the accuracy FastDTW is not effected very much by the characteristics or similarity of the input time series. FastDTW is significantly more accurate than the other two methods when the *radius* parameter is set to small values. When the *radius* parameter is larger, the abstraction method begins to approach the accuracy of FastDTW. However, FastDTW was always at least 2-3 times more accurate than abstraction in our experiments.

The three dotted Abstraction lines all have large errors for small *radius* values, but converge to less than 5% error on all data sets as the *radius* is increased to 30. This is due to the previously stated problem of the projected warp path being close to a near-optimal solution, but not taking local variations of the warp path into account. Abstraction does perform reasonably well if the *radius* is increased to at least 10. The ability of data abstraction to locally refine its projected path within the neighborhood of *radius* cells is not a part of the original algorithm, and is introduced in

this paper. The run-time of the original data abstraction algorithm is the same as our improved implementation when using a *radius* of 0, which has a very large average error of 983.3% over the three groups of data used in this evaluation.

The three dashed Band lines all have errors greater than 100% (as high as 7225%) for small *radius* values, and converge very slowly to 0% error as the *radius* increases. Band performs best on the random data because if two time series have almost nothing in common, an arbitrary warp path probably has a warp path distance that is not significantly much different from the minimum-distance or maximum-distance warp paths. The other two groups of data are data sets in a similar domain, which means that the optimal warp distance can be very small. Due to the way that error is calculated in Equation 14, if the optimal warp distance is very small, then the potential error can be very large because the optimal warp distance is the denominator of a fraction. The Band approach on the Trace data group has extremely poor accuracy because the time series contain events that are shifted in time, and bands only work well if a near-optimal warp path exists that is close to a linear warp. The Gun data group also does not work very well with the Band algorithm, which is surprising since the time series seem to be reasonably in phase with each other (near a linear warp).

4.2 Efficiency

4.2.1 Procedures and Criteria

The efficiency of the FastDTW algorithm will be measured in seconds, with respect to the length of the input time series, and compared to the standard DTW algorithm. The FastDTW algorithm will be run with the *radius* parameter set to: 0, 20, and 100 over a range of varying-length time series. The data sets used are synthetic data sets of a single period of a sine wave with Gaussian noise inserted. Only the lengths of the time series are significant because the shape of the time series has little significance on the run-time of either algorithm. The lengths of the time series evaluated vary from 10 to 150,000.

The standard DTW algorithm used in this evaluation is the linear-space implementation that only retains the last two columns of the cost matrix. If the standard DTW implementation is used, the test machine runs out of memory when the length of the time series exceeds ~3,000. The FastDTW algorithm is implemented as described in this paper except that the cost matrix is filled using secondary storage if the lengths of the time series grow so large that the number of cells in the search window is larger than can fit into main memory. Both algorithms are implemented in Java, and the runtime is measured using the system clock on a machine with minimal background processes running.

4.2.2 Results and Analysis

The FastDTW algorithm was significantly faster than the standard DTW algorithm for all but the smallest time series. FastDTW is 50 to 150 times faster than standard DTW (using *radius* values of 0 and 100 respectively) when the time series have lengths of 150,000 points. A sample of the results of the FastDTW algorithm can be seen in Table 2.

Table 2. Execution time (in seconds) of DTW and FastDTW on time series of four different lengths.

	Length of Time Series			
	100	*1,000*	*10,000*	*100,000*
DTW	0.02	0.92	57.45	7969.59
FastDTW (*radius*=0)	0.01	0.02	0.38	67.94
FastDTW (*radius*=100)	0.02	0.06	8.42	207.19

In Table 2, FastDTW and DTW have similar execution times for the 100 point time series. For the larger 10,000 and 100,000 point time series FastDTW runs much more quickly than DTW. But execution time for the 1,000 point time series is both faster and slower than DTW, depending on the *radius* parameter. The exact length at which FastDTW runs quicker than DTW depends on the *radius* parameter. Figure 10 shows the critical region where one algorithm is faster than the other depending on the *radius* parameter.

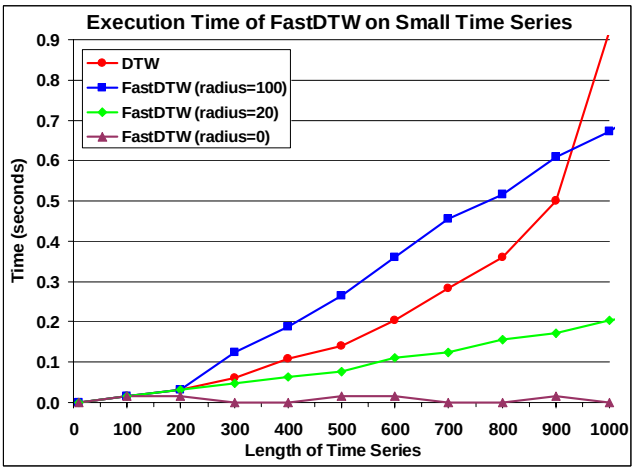


Figure 10. The efficiency of FastDTW and DTW on small time series.

The FastDTW algorithm, with a *radius* of 100, takes longer to run than DTW until the size of the time series exceeds approximately 900 points. However, with a *radius* of 0 or 20, the DTW algorithm is never faster than the FastDTW algorithm for small time series, and once the length of the time series exceed 200-300 points, FastDTW becomes the more efficient algorithm. For small time series it makes more sense to use the DTW algorithm rather than FastDTW. The FastDTW algorithm is not significantly faster (and possibly a little slower) than DTW for small time series, and DTW is guaranteed to always find the optimal warp path. However, for large time series, the quadratic time complexity becomes prohibitive.

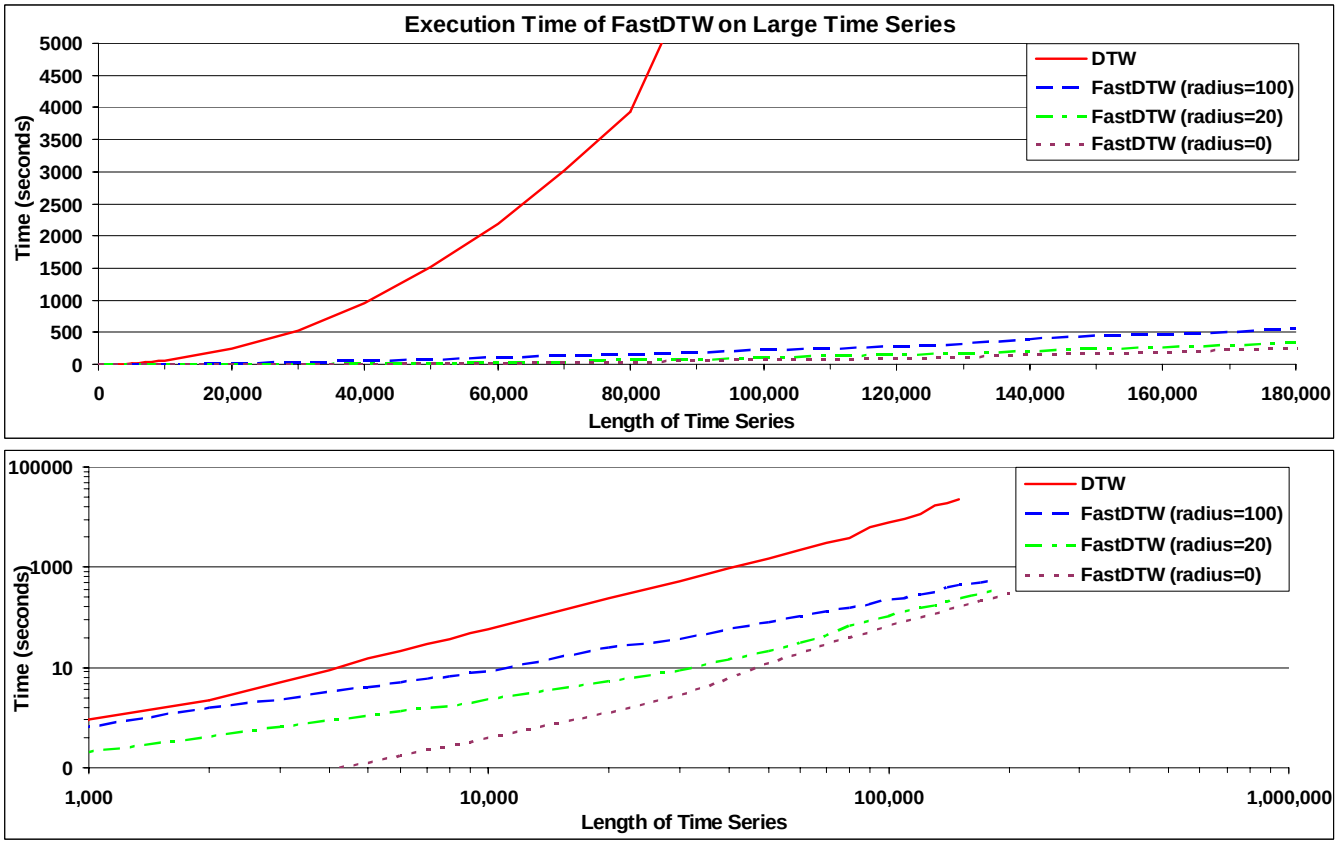


Figure 11. The eficiency of FastDTW and DTW on large time series. The top figure is scaled normally, and the bottom figure has log-log scaling.

The full results, using *radius* values of 0, 20, and 100 on time series ranging in length from 10 to 200,000 are shown in Figure 11. In Figure 11, the *y*-axis is the execution time and the *x*-axis is the length of the time series. The two graphs in Figure 11 are two views of the same data. The top graph is scaled normally, and the bottom graph has log-log scaling. Looking at the top graph, it is immediately obvious that the time complexity of DTW is much greater than that of FastDTW. DTW has an exponential curve, while all three FastDTW curves are approximately straight lines. In the log-scaled graph at the bottom of Figure 11, the three curves of FastDTW can be viewed more easily. The *radius* parameter increases the execution of FastDTW by a constant factor, which is why the three FastDTW lines seem to be converging on the log-scaled graph as length of the time series increases. The constant factor difference between them gets less significant as the length of the time series increases.

In Section 3.1, we proved theoretically that the FastDTW algorithm was O(*N*). Using the empirical data in Figure 11, the equation of the FastDTW curve with a radius of 100 is

$$y = 0.00000001x^2 + 0.001x - 0.7337$$

This coefficient of the squared term is very small, and it seems like the linear term is the most significant term in the equation...which would empirically prove that the FastDTW algorithm is O(*N*). However, since the values for *x* are so large, the squared term actually dominates the equation when *x*>100,000. The reason for this slight sub-linearity in the algorithm occurs when the number of cells being filled in the search window will not fit into main memory, and must be saved to the disk. Writing the cells to the disk can be performed in linear time. However, when reading the cells from the random-access file to construct a warp path, reading individual non-sequential cells from the disk cannot be performed in linear time. Larger time series create larger swap files, which require the disk head to move further to perform each random-access read operation. In other words, the number of cells in the cost matrix that must be filled/read is linear with respect to the length of the time series. So the *algorithm* is O(*N*), but the *implementation* is not quite O(*N*) for large time series when the entire search window will not fit into main memory.

5. CONCLUDING REMARKS

In this paper we introduced the FastDTW algorithm, a linear and accurate approximation of dynamic time warping (DTW). FastDTW uses a multilevel approach that recursively projects a warp path from a coarser resolution to the current resolution and refines it. While the quadratic time and space complexity of DTW has limited its use to only the smallest time series data sets, FastDTW can be run on much larger data sets. FastDTW is an order of magnitude faster than DTW, and it also compliments existing indexing methods that speed up time series similarity search and classification.

Our theoretical and empirical analysis showed that FastDTW has a linear time and space complexity. Expirical results have also

shown that FastDTW is accurate when warping both similar and dissimilar time series. With a *radius* of only 1, FastDTW had an average error of 8.6%, and increasing the *radius* to 20 lowers the error to under 1%. FastDTW's accuracy was compared to two existing methods, Data Abstraction and Sakoe-Chiba Bands, and was found to be far more accurate than either approach when using small *radius* values. FastDTW's solutions also always approached zero error (optimal warp path) with smaller *radius* values than the other two methods. An additional contribution of this paper is demonstrating how to apply the refinement portion of the FastDTW algorithm to the Data Abstraction approximate DTW algorithm. Doing so increased the accuracy of Data Abstraction by more than 100-fold in our evaluation with a *radius* of only 10.

The main limitation of the FastDTW algorithm is that it is an approximate algorithm and is not guaranteed to find the optimal solution (although it very often does). If for some reason a problem requires optimal warp paths to be found. Future work will look into increasing the accuracy of FastDTW. Possibilities to increase the accuracy of FastDTW include changing the step size (magnitute of the resolution change) between resolutions and evaluating search algorithms to guide search during the refinement step rather than simple expanding the search window in both directions.

6. REFERENCES.

[1] Abdulla, W., D. Chow, and G. Sin, Cross-words reference template for DTW-based speech recognition systems, in Proc. IEEE TENCON, Bangalore, India, 2003.

[2] Chu, S., E. Keogh, D. Hart & Michael Pazzani. Iterative Deepening Dynamic Time Warping for Time Series. In *Proc. of the Second SIAM Intl. Conf. on Data Mining*. Arlington, Virginia, 2002.

[3] Gupta, L., D. Molfese, R. Tammana & P. Simos. Nonlinear Alignment and Averaging for Estimating the Evoked Potential. In *IEEE Transactions on Biomedical Engineering*, vol. 43, no. 4, pp. 346-356, 1996.

[4] Itakura, F. Minimum Prediction Residual Principle Applied to Speech Recognition. In *IEEE Trans. Acoustics, Speech, and Signal Proc*. vol. ASSP-23, pp 52-72, 1975.

[5] Karypis, G., R. Aggarwal, V. Kumar & S. Shekhar. Multilevel Hypergraph Partitioning: Application in VLSI Domain. In *Design Automation Conl*, pp. 526-530. Anaheim, California, 1997.

[6] Keogh, E. Exact Indexing of Dynamic Time Warping. In *VLDB*, pp. 406-417. Hong Kong, China, 2002.

[7] Keogh, E. and T. Folias. The UCR Time Series Data Mining Archive [http://www.cs.ucr.edu/~eamonn/TSDMA/ index.html], Riverside, CA, University of California – Computer Science and Engineering Department, 2002

[8] Keogh, E. & M. Pazzani. Derivative Dynamic Time Warping. In *Proc. of the First Intl. SIAM Intl. Conf. on Data Mining*, Chicago, Illinois, 2001.

[9] Keogh, E. & M. Pazzani. Scaling up Dynamic Time Warping for Datamining Applications. In *Proc. of the Sixth ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pp.285-289. Boston, Massachuseetts, 2000.

[10] Kim, S., S. Park & W. Chu. An Index-based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases. In *Proc. 17th Intl. Conf. on Data Engineering*, pp. 607-614. Heidelberg, Germany, 2001.

[11] Kruskall, J. & M. Liberman. The Symmetric Time Warping Problem: From Continuous to Discrete. In *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*, pp. 125-161, Addison-Wesley Publishing Co., Reading, Massachusetts, 1983

[12] Ratanamahatana, C. & E. Keogh. Making Time-series Classification More Accurate Using Learned Constraints. In *Proc of SIAM Intl. Conf. on Data Mining*, pp. 11-22. Lake Buena Vista, Florida, 2004.

[13] Sakoe, H. & S. Chiba. (1978) Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans. Acoustics, Speech, and Signal Proc., Vol. ASSP-26.

[14] Salvador, Stan. Learning States for Detecting Anomalies in Time Series. Master's Thesis, CS-2004-05, Dept. of Computer Sciences, Florida Institute of Technology, 2004.

[15] Vlachos, M., G. Kollios, & D. Gunopulos. Discovering Similar Multidimensional Trajectories. In *Proc. 18th Intl. Conf. on Data Engineering*, pp. 673-684, San Jose, California, 2002.

Finding Reference Affinity Groups in Trace Using Sampling Method

Chengliang Zhang, Yutao Zhong, Chen Ding, Mitsunori Ogihara
Computer Science Department
University of Rochester
Rochester, NY 14627-0226
{zhangchl,ytzhong,cding,ogihara}@cs.rochester.edu

ABSTRACT

Making better use of the cache is important for the modern computer programs and systems. One key step is understanding the data locality. In this article, we investigate one effective and important model of data locality—reference affinity. This model is superior to previous ones because it can express whole-scale locality in a more accurate and flexible way. Traces collected from different applications are a rich source for the analysis of reference affinity. In this article, we extend strict reference affinity to weak reference affinity and prove their properties. We propose a sampling method to find reference affinity groups and present experimental results based on synthetic data showing that the new method is more scalable and accurate than the state-of-art method proposed by Zhong *et al.* [29].

Keywords: sequence data mining, reference affinity, sampling method, reuse distance

1. INTRODUCTION

Analyzing reference affinity is an important task for improving performance of programs, storage systems and Internet systems. Because of the huge performance and cost gaps [11] between CPU, memory and disk, cache is widely used in the modern computer systems. However, achieving good cache utilization is difficult. It depends highly on how much we understand the locality in programs and systems, and how we make use of this locality. In particular, Reference affinity is an effective way of expressing the long-range locality.

The locality of programs and systems can often be predicted from their past behaviors. Here we call the past behavior a *trace*. Traces may have different meanings in different application domains. For example, in program analysis, by profiling technique, we can get the sequence of memory accesses for selected data inputs. In a storage system, we can record the sequence of blocks that are accessed. In an Internet system, the web log records file accesses. In real applications, the accesses of data elements take various lengths of time. In locality analysis, each access is assigned a logical time, because its relative order matters much more than its access time.

Most of previous work focus on finding repeated patterns from a trace. The techniques of finding repeated patterns have been extensively studied [18; 27; 4]. For example, Li *et al.*used frequent sequence mining to find the correlation between blocks and applied that information for the data prefetching and data layout [16]. However, there are some drawbacks in expressing the locality using repeated patterns. First, it fails to express repeated cases within the patterns. For example, in sequence 1, A, B and C are not repeated patterns, but they apparently have strong locality. Secondly, it has difficulty considering the small variations within the patterns. For example, in sequence 2, it is hard to detect that A, B and C have strong locality. Thirdly, locality analysis ignores the order of the access, which is critical for sequential patterns. So the frequent sequence mining is not adequate. An example in Sequence 3 explain this clearly. In this article, we use ellipse to denote the data elements other than the ones shown in the sequence.

$$\ldots[ABBBC]\ldots[ABC]\ldots[ABCC]\ldots \quad (1)$$

$$\ldots[ADBEC]\ldots[AFBC]\ldots[ABGC]\ldots \quad (2)$$

$$\ldots[ABC]\ldots[ACB]\ldots[BAC]\ldots \quad (3)$$

Strict reference affinity measures how closely a group of data elements are accessed together in one trace. Using the concept of strict reference affinity group, we can get a hierarchical partition of the data elements [29]. By using the reuse distance [6], originally called LRU stack distance [19], it overcomes the shortcomings mentioned previously and thus expresses the locality of data in a more accurate and appropriate way. In general, the reference affinity analysis problem is NP-hard and thus has no known efficient solution. Zhong *et al.* [29] proposed an necessary but not sufficient approach using k-distance analysis based on reuse distance. Strict reference affinity is very successful in program optimization [28]. Many programs contain a large number of homogeneous objects. Examples include particles in a physical system and molecules in a biological system. In Fortran 77, attributes of an object are stored separately in arrays. In languages such as C and C++, attributes of an object are stored together in a structure. Both schemes are fixed and neither is sensitive to the access pattern of a program. Reference affinity allows a compiler to group attributes that are accessed together. For Fortran programs, the transformation is array regrouping; for C and C++ programs, it is structure splitting. Zhong *et al.*showed that their k-distance analysis consistently outperforms data organizations given by the programmer, compiler analysis, frequency profiling, and statistical clustering. It shows that the performance

impact of array regrouping and structure splitting is on average 16% on an IBM server and 20% an Intel PC, and the impact is greater on newer faster machines.

k-distance analysis, however, may cluster irrelevant data elements into one group. This occurs especially when the reuse patterns are very similar,such as when the single data element appears randomly. The concept of strict reference affinity also hampers the analysis of looser groups such as those appearing in the web logs.

In this paper, we extend the model of affinity group by considering both strict reference affinity and weak reference affinity. We also present a sampling method to find the interesting reference affinity groups. Compared with the method proposed by Zhong *et al.*, this method requires scanning only a small portion of the trace to guarantee finding a "good" solution. This is very important when the trace is too large to be processed even once. This method can also distinguish between groups with similar reuse signatures better. Experiments show that the sampling method performs much better than the k-distance analysis in term of both accuracy and scalability.

The rest of this paper is organized as follows. In Section 2, we review related concepts, extend strict reference affinity to weak reference affinity and discuss their properties. Section 3 presents the sampling method, its properties and choices of thresholds. As a comparison, we review the k-distance analysis based on reuse signature. In Section 4, we design a synthetic trace generator and present an extensive experimental comparison between the sampling method and k-distance analysis. Related work are reviewed in Section 5. We conclude with a summary and directions for future work in Section 6.

2. REFERENCE AFFINITY GROUP

The concept of trace-level reference affinity was first proposed by Zhong *et al.* [29]. Here we review the concept of strict reference affinity and generalize it to weak reference affinity. Further, we prove some interesting properties that hold for weak reference affinity.

For concreteness, we first define a data element space:

Definition 1. **Data Element Space.** A data element space is a finite set of data elements to be investigated and it should include all of the data elements that appears in a trace. The data element could be a memory location, a file or a disk block. We can use categorical symbols to represent data elements. In this article we use S to denote the data element space and symbol such as x, y, A, B to denote a specific data element.

Then we define a trace:

Definition 2. **Trace.** A trace is an ordered sequence of data elements that was accessed. If we assign a logical time to each access, then trace can be deemed as a many-to-one function π mapping from logical time to data elements. $\pi(t) \in S$ is the data element that was accessed at logical time t, where t ranges from 1 to T and T is the length of the trace.

Correspondingly, we can define an inverse function for π mapping from the data element space to a subset of logical times.

$$\Gamma(x) = \{t|\pi(t) = x\}. \tag{4}$$

Following the definition of reuse distance in [6; 19], we can define the reuse distance as:

Definition 3. **Reuse Distance.** Reuse distance between two logical time i and j is the number of distinct data elements accessed between i and j. Formally, we define it as a function δ:

$$\delta(i,j) = \begin{cases} |\{\pi(t)|j \le t < i\}| & \text{if } i > j, \\ |\{\pi(t)|i \le t < j\}| & \text{if } i < j, \\ 0 & \text{if } i = j. \end{cases} \tag{5}$$

where $1 \le i, j \le T$ and "$|Q|$" denotes the size of the set Q.

Reuse distance is a metric. Since reuse distance is the volume of data accessed between two logical time, we also call it volume distance. We want to use reuse distance instead of the simple time distance, because from the study of Chen and Zhong [6], reuse distance can identify consistent patterns in the presence of dynamic data allocation and input-dependence control flow and thus it reveals the program patterns of locality more precisely. A simple example may explain this clearly. Consider the case in sequence 6. Using only the time difference, it seems that A, C and D, H are equally closely referenced. However, using reuse distance, we know that A, C are more closely referenced since their reuse distance is 2, which is smaller than the reuse distance between D and H, which is 4.

$$\ldots ABBBC \ldots DEFGH \ldots \tag{6}$$

Before we go to the concept of reference affinity, let us define the k-linked path relative to a group of data elements.

Definition 4. **k-linked Path Relative to Group G.** We say there is a k-linked unidirectional path between logical time i and j (without loss of generality, let us suppose $i \le j$) relative to a group G of data elements if and only if there exist a list of logical time $t_1, t_2, \ldots, t_n$, such that:

1. $i < t_1 < t_2 < \ldots < t_n < j$;
2. $\pi(i), \pi(t_1), \pi(t_2), \ldots, \pi(t_n), \pi(j)$ are distinct and belong to the set G;
3. $\delta(i, t_1) \le k \wedge \delta(t_1, t_2) \le k \wedge \ldots \wedge \delta(t_n, j) \le k$.

It is clear that if there is a k-linked path relative to group G for i, j, then for $k' > k$, there is a k'-linked path relative to group G for them. Figure 1 gives an example of 2-linked path between 1 and 8 relative to group $G = \{ACDFH\}$. We also say that 1 and 8 are 2-linked relative to G.

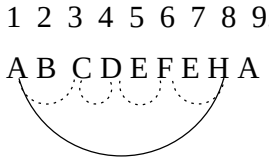


Figure 1: K=2, G={A,C,D,F,H}, i=1, j=8

The data elements can be grouped together based on how close they are accessed in a trace. Here we use a formal definition to capture the sense of closeness.

Definition 5. **Strict Reference Affinity Group.** Given a trace and k, a group G of data elements is a strict reference affinity group if and only if

1. for any x and y in G and for any $i \in \Gamma(x)$, there exists a $j \in \Gamma(y)$, where i and j are k-linked relative to the group G.
2. There does not exist G', such that $G \subset G'$ and G' also satisfies condition (1).

Strict reference affinity groups have important properties, as shown in [29].

THEOREM 1. *Given a trace and a link length k, strict reference affinity groups form a unique partition of the data element space.*

THEOREM 2. *Given a trace and two link length k and $k'(k < k')$, strict reference affinity groups at k from a finer partition of the affinity groups at k'.*

THEOREM 3. *Given a trace and a strict affinity group G at link length k, for any logical time t where $\pi(t) \in G$, then there exists a section of the trace that covers t and at least one access to all other members of G. The volume distance between the two sides of the section is no greater than $2k|G|+1$.*

In many cases, however, strict reference affinity is too strict to allow any useful groups. We give a weaker version of the reference affinity.

Definition 6. **Weak Reference Affinity Group.** Given a trace, parameter k, and threshold θ, a group G of data elements is a weak reference affinity group if and only if

1. for any $x, y \in G$,
   - either $\frac{|\{i|i\in\Gamma(x)\wedge\exists j\in\Gamma(y)\wedge\ i,j \text{ are } k\text{-linked relative to } G\}|}{|\Gamma(x)|} \geq \theta$; (We call x and y are weakly directly connected in this case.)
   - or there exists distinct $z_1, z_2, \ldots z_n \in G$, such that x and z_1, z_1 and z_2, ..., z_n and y are weakly directly connected.
2. There does not exists G', such that $G \subset G'$ and G' also satisfy condition (1).

When θ is bigger than 1 or smaller than 0, weak reference affinity is trivial, so we consider only the case $0 \leq \theta \leq 1$. k is similarly bounded by the 1 and the size of data element space $|S|$, since we use reuse distance.
The following theorem shows the relationship between strict reference affinity and weak reference affinity.

THEOREM 4. *Given a trace and distance k, strict reference affinity groups form a finer partition of weak reference affinity groups at threshold θ.*

PROOF. Actually, we just need to show that any strict reference affinity group G is a subset of one weak reference affinity group. This is equal to showing that data elements in the same strict reference group G will remain in the same weak reference group.
Actually, we know that for any two data elements x, y in G, for every $i \in \Gamma(x)$, there exists a $j \in \Gamma(y)$, such that i, j are k-linked relative to group G. So, $\frac{|\{i|i\in\Gamma(x)\wedge\exists j\in\Gamma(y)\wedge\ i,j \text{ are } k\text{-linked relative to } G\}|}{|\Gamma(x)|} = 1 \geq \theta$, which means that x and y are weakly directly connected in G. So group G satisfies condition (1) of weak reference affinity. New data elements can be added into this group until it is a weak reference group. □

However, we should notice that even when θ is 1, weak reference affinity groups are not exactly strict reference affinity groups. For example, in sequence 7, $\{X, W, Y, Z\}$ belongs to the same weak reference affinity group with threshold being 1 and k being 2. However, they belong to three strict reference affinity groups:$\{X\}, \{WY\}, \{Z\}$. We use affinity groups to denotes both strict and weak reference groups in the rest of the paper, except when explicitly mentioned.

$$\ldots XAWBYCWDZ \ldots XEWFYGWHZ \ldots \quad (7)$$

Similarly with theorem 1, 2 and 3, we can prove the following properties for weak reference affinity.

THEOREM 5. *Given a trace, a link length k and a threshold θ, weak reference affinity groups form a unique partition of data element space.*

PROOF. We just need to show that any data element belongs to one and only one weak reference affinity group.
The first part is straightforward. For any data element x, the singleton group $\{x\}$ of course satisfies the first condition of definition 6. Then we keep adding to this group new data elements which do not break the condition (1). Because there is a finite number of data elements, we get a weak reference affinity group which includes x.
We prove the second part by contradiction. Suppose the two weak reference affinity groups are G_1, G_2 and $G_1 \neq G_2$. They have common data element x. Now let's look at the group $G' = G_1 \cup G_2$.
For any two data elements $y, z \in G'$. If both of them belong to G_1 and G_2, then clearly condition (1) holds. Without loss of generality, assume $y \in G_1 \wedge y \notin G_2$ and $z \in G_2 \wedge z \notin G_1$. Since $x \in G_1$, so either x is weakly directly connected with y or there exist $v_1, v_2, \ldots v_m \in G_1 \subset G'$, such that $x, v_1, v_2, \ldots, v_m, y$ are weakly directly connected one by one. Similarly, either x are weakly directly connected with z or there exists $w_1, w_2, \ldots w_n \in G_2 \subset G'$, such that $z, w_1, w_2, \ldots, w_n, x$ are weakly directly connected. Then actually, $z, \cdots, x, \cdots, y$ are weakly directly connected. If there are repeated data elements in this sequence, we just need to cut out the data elements between them. We can see that y and z also satisfies condition (1).
Since clearly, $G_1 \subset G'$ and $G_2 \subset G'$, and G' also satisfies condition (1). This contradicts with the fact that G_1, G_2 are weak reference affinity groups. Therefore, every element belongs to only one weak affinity group. □

THEOREM 6. *Given a trace and two link lengths k and $k'(k < k')$, weak reference affinity groups at k form a finer partition of the affinity groups at k', given the same threshold.*

PROOF. Since if i, j are k-linked relative to group G, then they are also k'-linked relative to group G, G satisfies condition (1) when k changes to k'. Thus all of the data elements

in group G at k remain at the same group at k'. So G must be a subset of some affinity group at k', that is, weak reference affinity groups at k forms a finer partition of the affinity groups at k'. □

THEOREM 7. *Given a trace and two threshold θ and $\theta'(\theta < \theta')$, weak reference affinity groups at θ form a finer partition of the affinity groups at θ', given the same link length k.*

The proof is very similar with the previous theorem, so we will not go into detail here.Following the previous two theorems, we can immediately get the following theorem.

THEOREM 8. *Given a trace, weak reference affinity groups corresponding to different k and θ form a lattice structure.*

We restrict $0 \leq \theta \leq 1$ and $1 \leq k \leq |S|$. We can express weak reference affinity groups corresponding to k and θ as $< k, \theta >$. By theorem 6 and 7, it is easy to see that different $< k, \theta >$ form a lattice structure.
Theorem 3 does not hold for weak reference affinity groups any more. For example, suppose the trace is $ABDEFGHABIABJABDEFGHA$. let θ be 80% and $k = 2$. We get the following weak reference affinity groups: AB, DEFGH, I, J. But apparently, there does not exist such a section with size no great than 5 for the last occurrence of A. However, we have the following verification theorem.

THEOREM 9. *Given a trace and a weak affinity group G at link length k and threshold θ, for any $x \in G$, there exists a $y \in G$, such that there are more than $|\Gamma(x)| \times \theta$ sections of trace such that:*

1. *these sections include x and y at the two ends;*
2. *the volume distance of every section is within $k(|G| - 1) + 1$.*

PROOF. Since $x \in G$, then there exists at least a different data element $y \in G$, such that x and y are weakly directly connected. From the definition of weak reference affinity group, we have $\frac{|\{i|i\in\Gamma(x)\wedge\exists j\in\Gamma(y)\wedge\ i,j \text{ are } k\text{-linked relative to } G\}|}{|\Gamma(x)|} \geq \theta$. This is equivalent to saying that there exists more than $|\Gamma(x)| \times \theta$ of logical time i, where $\pi(i) = x$ and for each such i, there exists a $j \in \Gamma(y)$, which makes i and j k-linked relative to G. Since i and j are k-linked relative to G, then based on the definition of k-linked, we know that the subsequence of the trace from i to j is just the section of trace with x and y at the two ends and its volume distance is within $k(|G| - 1) + 1$, since the reuse distance is Euclidean. □

The following two concepts describe how we are interested in the affinity groups we found.

Definition 7. **Naive Reference Affinity Group.** A reference affinity group is naive if and only if it includes only one data element.

Definition 8. **Interesting Reference Affinity Group.** Given a trace and threshold η, we say a reference affinity group G is interesting if and only if the group is not naive and every data element in the group appears more than η times in the trace.

3. A SAMPLING APPROACH

In this section we propose a sampling method for reference affinity problem. We start by giving some basic definitions. Then we describe the sampling method and prove its properties. We compare it with the k-distance analysis based on reuse signature. Deciding the appropriate thresholds is very important for both sampling method and k-distance analysis. We also address some possible solutions here.

3.1 Sampling algorithm

Definition 9. **Verification Section.** Given a trace, an affinity group G and $x, y \in G$, if there is a section of the trace covering both logical t, where$\pi(t) = x$, and another data element y, and the length of the section is no greater than $k(|G| - 1) + 1$, then we call it a verification section for x and y. Among those verification sections for t and y, the shortest one is called the critical verification section from x to y.

Verification sections witness the existence of affinity from x to y. For strict reference affinity group, there are at least $|\Gamma(x)|$ critical verification sections from x to y. As for weak reference affinity group, for every x, there exists at least a y in the same group, such that there are more than $|\Gamma(x)| \times \theta$ critical verification sections from x to y.

Definition 10. **Sliding Window.** Given a trace and a length n, a sliding window of size n is a section of trace with volume distance n.

Our sampling method (algorithm 1) first estimates the upper bound of size of the affinity groups. Suppose it is g. The sample size is accordingly defined as $l = 2gk$. The sampling method then considers the confidence for x, y, *i.e.*, how many percents of the sampled sliding windows having both x and y among the sampled sliding windows that x or y appears, whichever is smaller. If the confidence value is bigger than some threshold θ', then x and y are claimed to be weakly connected. Weakly connected data elements are then clustered as one group. The idea here is a kind of similar with association rule mining [8].
Now let's consider interesting reference groups. Let δ be the sample rate, η be the interesting threshold. Then every data element in the interesting group is sampled at least $l\delta\eta$ times, if the data elements are randomly scattered in the trace. In our sampling method, we allow user to specify the threshold ϵ.

ALGORITHM 1. Sampling method for reference affinity groups **Input:** *A trace; sample size l; sample rate δ; threshold ϵ; Affinity threshold θ'.*
Output: *the interesting reference groups.*
method:

Let W be the number of sampled sliding windows each data element appears and M be the number of sampled sliding windows where each pair of data elements appears.
Sample sliding windows of size l from the given trace according to the sampling rate δ.
for** each sliding window **do
  for** distinct data element x **do
    increase $W[x]$ by 1.
  end for

for every distinct pair of data element x, y *do*
increase $M[x, y]$ by 1
end for
end for
ignore those data elements s with $W[s] < \epsilon$.
Construct a graph with the data elements not ignored as vertices.
for two vertices x, y *do*
if $confidence(x, y) \equiv M[x, y]/min(W[x], W[y]) > \theta'$ *then*
add an edge between x and y
end if
end for
Output every connected subgraph as a group. **end**

THEOREM 10. *For any data element x in the reference affinity group at threshold θ, there exists a y in the same group and their expected confidence is greater than $\frac{\theta}{2}$.*

PROOF. Suppose the upper bound for the affinity group size is g. Consider reference affinity group G. Clearly, $|G| \leq g$. For any data elements $x \in G$, from the definition of weak reference affinity group, we can find a $y \in G$ which is weakly directly connected to it. Let's consider the critical verification sections for them. For every critical section, we know its volume size is within $k(|G| - 1) + 1$. The sample size is $2kg$. So the sliding window have

$$1 - \frac{k(|G| - 1) + 1 - 1}{2kg} \geq 1 - \frac{k|G|}{2kg} \geq 1 - \frac{kg}{2kg} = \frac{1}{2}$$

probability of covering x and y given it covers x.
Since x, y are weakly directly connected, there are at least $|\Gamma(x)|\theta$ corresponding critical verification sections. Therefore, the confidence for x and y is at least

$$\frac{|\Gamma(x)|\theta\frac{1}{2}\delta}{|\Gamma(x)|\delta} = \frac{\theta}{2}.$$

□

By theorem 10, we know that if we set the threshold θ' to be $\frac{\theta}{2}$, we can ensure that the data elements in the same weak reference affinity group remain in the same group found by algorithm 1.
Now, let us consider the time and space complexity of this algorithm. Suppose N is the size of data element space, L be the length of the trace. The time complexity of this algorithm is $O(L\delta N^2)$. The space complexity is $O(N^2)$, since we need to store pairwise information. People can refer to [22] for more efficient sampling algorithms.
The sampling method has two shortcomings. First, it need an upper bound for the group size. In some cases, the choice of group size is too wide to get a good estimation. For example, in web log analysis, there are generally tens of thousands files. We will analyze how this parameter impacts the affinity group analysis in Section 4. Secondly, the memory requirement is $\frac{N(N-1)}{2}$. However, in real applications, the combinations of pairs occur much fewer than $\frac{N(N-1)}{2}$.

3.2 Comparison to k-distance Analysis

Let us first review how the k-distance analysis based on reuse signature [29]. Remember that this algorithm only targets strict reference affinity groups.

It first measures the reuse signature for every data element. The reuse signature of a data element is the histogram of the reuse distance of their accesses. For example, Figures 2 through 4 show the space-time graph of three fields of the tree node structure in *Cheetah* with a simple input. Figure 5 shows the reuse signature of the three fields with reuse distance greater than 1024. The x-axis is a sequence of bins in the log-2 scale. The y-axis gives the number of memory accesses whose reuse distances fall into each bin. We can see from figure 5 that fields *rtwt* and *lft* have very similar access patterns, but they differ from the field *addr*. This is right the information we get from Figures 2 through 4. Thus, reuse signature is a good representation of the reuse patterns. After this step, accesses to every data element is summarized into a vector, each representing the number(or average, or percentile) of reuse distances in the corresponding bin.

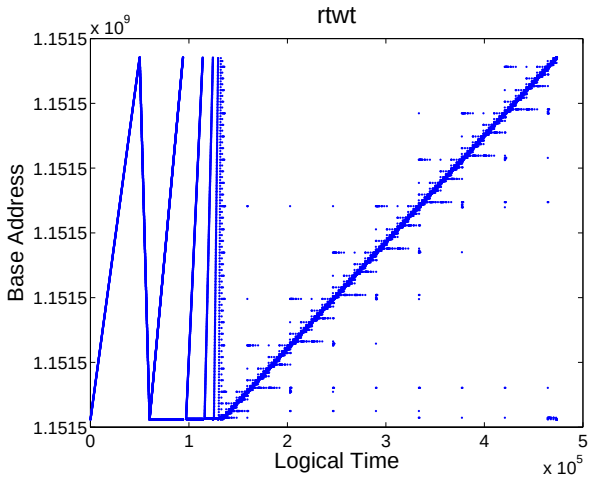


Figure 2: Space-time graph of the accesses of *rtwt*

Then it computes the Manhattan distance between every two data elements x, y as:

$$d(x, y) = \sum_{i=1}^{B} |Avg_i^x - Avg_i^y| \quad (8)$$

where Avg_i^x is the average distance of data elements in bin i, B is the number of bins considered. Careful analysis showed that $|Avg_i^x - Avg_i^y|$ is smaller than k for every bin [29]. So if $d(x, y) \leq k * B$, then x, y are clustered into the same affinity group.
Compared with the sampling method, this method tends to cluster irrelevant data elements into one group, especially for single data elements which occur randomly. Another problem is that the vector may not be the same for data elements in the same strict reference affinity group because of the partition boundaries, as we can see in figure 5. Considering that the histogram is presented in log scale, the difference between the average reuse distances of each bin may be bigger than k because of partition boundaries. The most serious problem for this method is that it does not work for weak reference affinity groups. Because data ele-

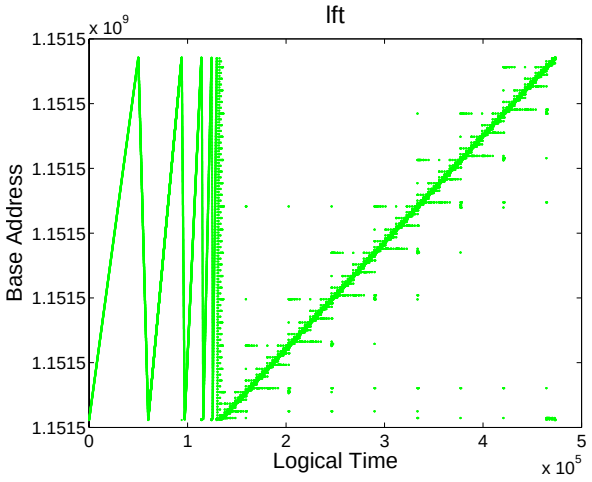


Figure 3: Space-time graph of the accesses of *lft*

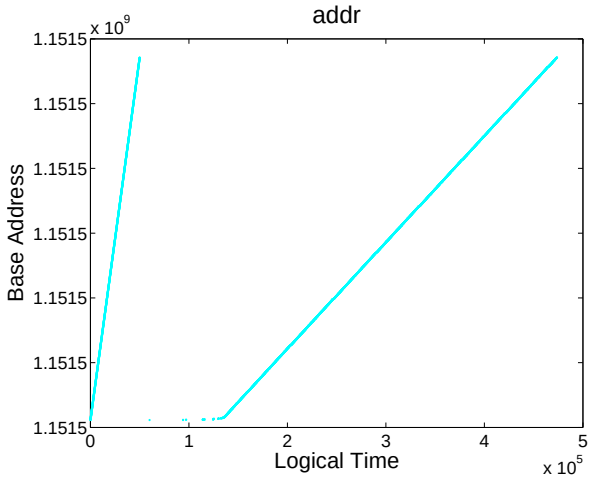


Figure 4: Space-time graph of the accesses of *addr*

ments in the same weak reference affinity groups may have very different reuse signatures.

3.3 Deciding the thresholds

In both the sampling method and k-distance analysis, an important question is deciding the threshold. Though proofs show that we can use $\frac{\theta}{2}$ for the sampling method and kB for the k-distance analysis, in practice, these values are always too loose to get enough useful affinity groups.

One way of solving this is to first select a series of thresholds and construct a hierarchical structure, which can be displayed using visualization tools. Then one can pick the best threshold manually. These thresholds for the hierarchical structure can be selected using statistical method such as the percentile of the edges of minimum spanning tree for k-distance analysis or maximum spanning tree for the sampling method. We use MST to denote them in both cases.

An automatic way of selecting the threshold is by choosing a value based on the edge distribution of the MST. From our empirical analysis, the edges of MST of k-distance analysis are exponentially distributed after sorting. An example is shown in figure 6. For MST of sampling method, they are distributed in a quadratic scale. So the idea is to change the curve to linear scale by taking the logarithm and then picking the midpoint of the line, which is a good estimation for most of our analysis. In our performance comparison, we use this strategy for both methods.

4. EVALUATION

In this section, we compare the sampling method with k-distance analysis method. We use artificial traces here instead of real application data because we want more control on the different factors that may affect the performances of different methods. We point out the potential applications in Section 6.

Programs for sampling method and k-distance analysis based on reuse signature are both implemented using C++ and compiled using g++. The synthetic trace is generated using a PERL program. All of our experiments are performed on a P4 2.8GHz machine with 1024M memory running Linux 2.4.20-19.9.

4.1 Synthetic trace

The generation of the synthetic trace is controlled by many parameters. Here we simplify the model of trace by using only the parameters shown in table 1. The generator first initializes the length and frequency of each affinity groups according to the use-specific type LT and LF. For example, if LT is set to be "linear", then the length of each affinity group increases monotonically, in our experiment, by 1. If LT is set to be "constant", then every affinity group is set to the same length, in our experiment, 10. LF has the similar meaning with LT. When LF is "linear", the frequency of each affinity group increases linearly between two ends, in our experiment, 0.3 to 0.8. When LF is "constant", then the frequencies are set equally to be a number, in our experiment, 0.7. Then the generator selects a non-naive affinity group randomly. Based on the frequency of that group, it decide to use it or another naive group to generate a section of the trace. We can see that the frequency is actually a relative frequency. If one non-naive group is selected, a random permutation is first done and then based on the weakness, a part of data elements are thrown away. The rest of the data elements are separated using the random naive data element based on K. The generator repeats the whole process until the size of the trace reaches T.

Note that in real applications, the situation is much more complex than described here. For example: the different non-naive reference affinity groups may overlap with each other; One occurrence of an affinity group may include multiple occurrence of each data element; One affinity group may be active in one period but silent in another.

4.2 Evaluation Criteria

In our experiments, we use two sets of metrics to compare the two methods described in Section 3.

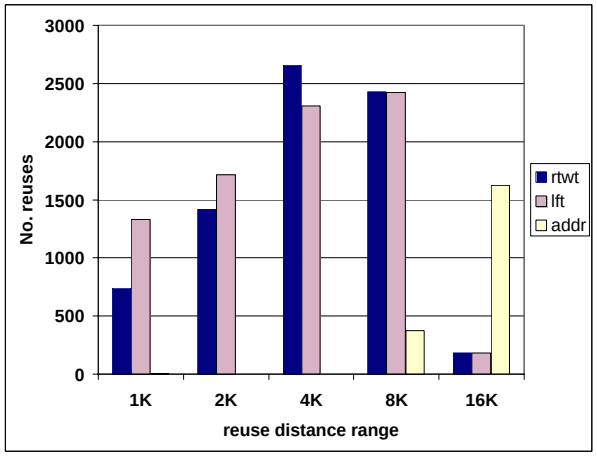


Figure 5: The reuse signatures of the three fields

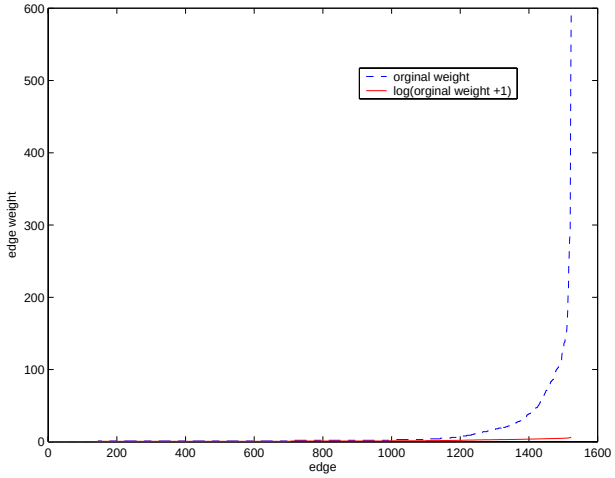


Figure 6: The distribution of the sorted weights of MST got by k-distance analysis

NA	Number of non-naive affinity groups
NN	Number of Naive affinity groups
T	Total size of the trace
LT	Type of length of affinity groups
LF	Type of the frequency of the affinity groups
W	Weakness of the affinity groups
K	k-affinity group

Table 1: Parameters for the artificial trace

The first one is direct and simple. We compare the affinity groups we used to generate the trace with the affinity groups found by the algorithm and use the perfectly matched affinity groups over the total number of groups as the accuracy. However, a more complex measure is desired. Suppose the "good" group is G. It is separated into parts $P_1, P_2, \ldots, P_n$ and scattered into the algorithm-detected groups $G_1, G_2, \ldots, G_n$. Then we can define the accuracy for this affinity group as:

$$accuracy(G) = \frac{\sum_{i=1}^{n} \frac{|P_i|}{|G_i|}}{n^2}.$$

The accuracy measurement reflects the loss of information. The more pieces a group is separated into, the more information we lose. If one part is clustered in one bigger group, we lose more. Here we want to use n^2 instead of n as the denominator because, considering the following situation: If G is scattered into exactly $|G|$ naive groups, then accuracy would be 1, if we use n as the denominator. At last, the whole accuracy is defined as the mean of the accuracies of affinity groups.

To distinguish between the two metrics, we call the first one as *match rate* and the second as *accuracy*. When the affinity groups are correctly recognized, both the match rate and the accuracy are 1. However, in general cases, the accuracy is higher than the match rate.

4.3 Performance Comparison

In this section we present our performance comparison from four different perspectives. Every experimental result presented here is an average of 20 times simulation. Since for all of the experiments, variances of the accuracy and miss rate are very small, we will not present them in the figures. Table 2 shows the parameter setting for each experiment. For every experiment, there is 200 naive affinity groups. Asterisks denote that values that are not fixed and will be explained during the comparison. In the table, "L" denotes the linear type, "C" denote the constant type. The size of the trace is 200000 by default. If the LT is set to linear, then the size of trace is set to ensure that every affinity group occurs roughly 400 times. S column is added for the sample size.

Figure	NA	LT	LF	W	K	S
7	20	L	C	1	1	*
8	*	L	C	1	1	*
9	*	C	L	1	1	30
10	*	L	C	1	1	102
11	50	C	L	1	*	30
12	50	L	C	1	*	30
13	*	C	L	1	1	2*NA
15	*	L	C	1	1	30
14	*	C	L	0.7	1	2*NA
16	*	L	C	0.7	1	30

Table 2: Parameters for the artificial data

In the first set of experiments, we analyze how the sample size affects the accuracy. Figure 7 shows the performance of different sample sizes for 20 strict non-naive 1-affinity groups with a linear length. The best performance is achieved when the sample size is 20, which is comparable to kg. The reason is that in our experiment, the data elements are randomly scattered, so the average distance is about half of kg. Thus using only kg, we can get enough confidence to cluster the

data elements together. We should pay attention that the sampling method is not very sensitive to the choice of the sample size. When the sample size is between 10 and 30, the performance changes very little. Even from 5 to 50, the performance is within 8%, which means fewer than two incorrect affinity groups. Figure 8 shows the performance using the sample size $2kg$, labeled flexible sample size in the figure. In the mean time, we use the fixed sample size 30 as an comparison. Though not optimal, results shows that $2kg$ is a relatively good size, meaning usually it can get more than 90% accuracy. Between small and big estimations of g, we prefer the latter. Clearly, since when the g is far underestimated, the performance curve drops down quickly.

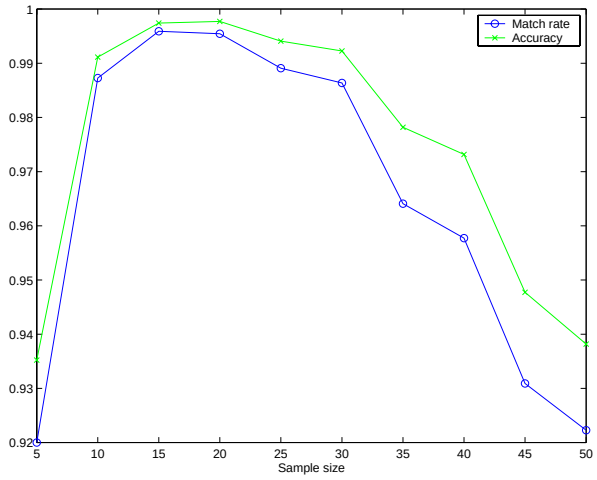


Figure 7: The effect of sample size

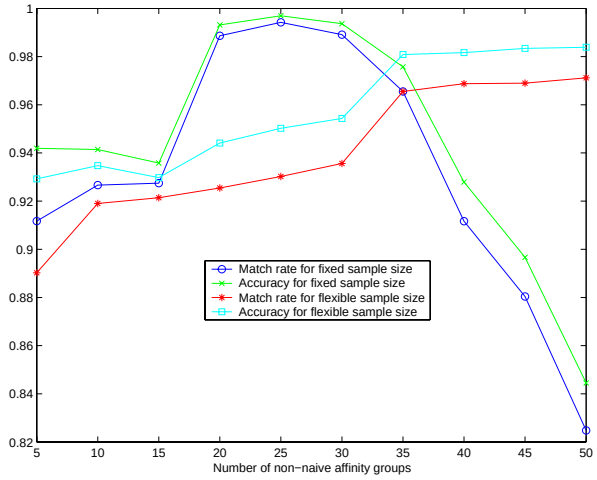


Figure 8: The effect of sample size

The second set of experiments compares the methods using groups of different lengths and frequencies. Figure 9 shows the performance comparison on affinity groups of various frequencies. Both methods perform well in this case. The sampling method can almost always find all of the groups correctly, which is much better than k-distance analysis. When the weakness changes from 0.5 to 1, both methods improve their ability of recognizing the groups in a linear scale. Figure 10 shows the performance comparison on affinity groups of various lengths. When the weakness increases, the performance of sampling method is stable. However, the performance of k-distance analysis drops. The reason is that when weakness is 0.5, this algorithm just cluster every data element into a trivial group. When the weakness becomes larger, it makes more errors since it clusters the naive affinity groups together. Despite this, the match rate for non-naive groups increases and the overall performance drops .

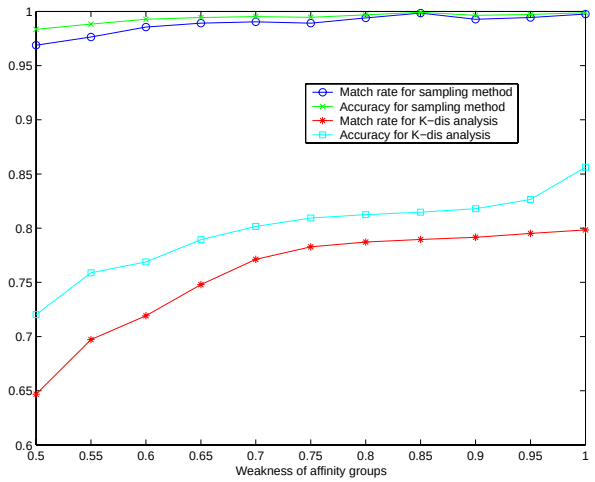


Figure 9: Performance comparison based on weakness of affinity groups having various frequency

In the third set of experiments, we compare the effect of k on both algorithms. From both figures 11 and 12, we can conclude that when k increases, the complexity of finding affinity groups increases. The performance of the sampling method is more stable than that of k-distance analysis, which drops down very fast when k is bigger than 3. The reason is that when the k increases, the reuse signatures become more random. This raises the affinity threshold and leads to more affinity groups being clustered together.

The fourth set of experiments compares the scalability of both algorithms in finding strict and weak reference affinity groups. Figures 13 and 14 show the performance by increasing the number of affinity groups of fixed length and linear frequencies. Apparently, the performance of the sampling method scales better than that of k-distance analysis. The performance of k-distance analysis first increases, then decreases, because when the number of groups is small, the reuse signature of every data element is close and k-distance analysis just clusters every data element as one

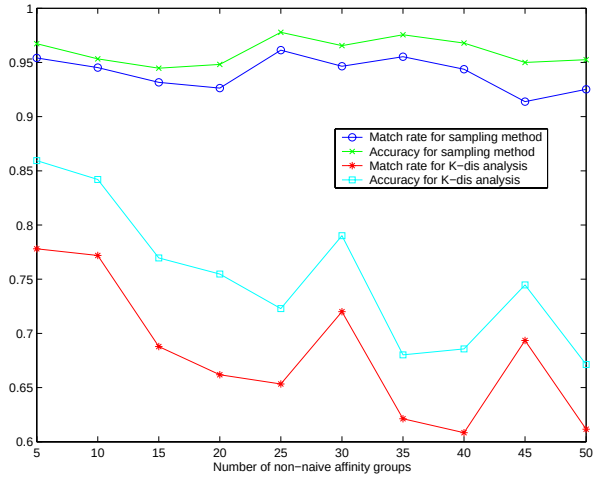


Figure 10: Performance comparison based on weakness of affinity groups having various length

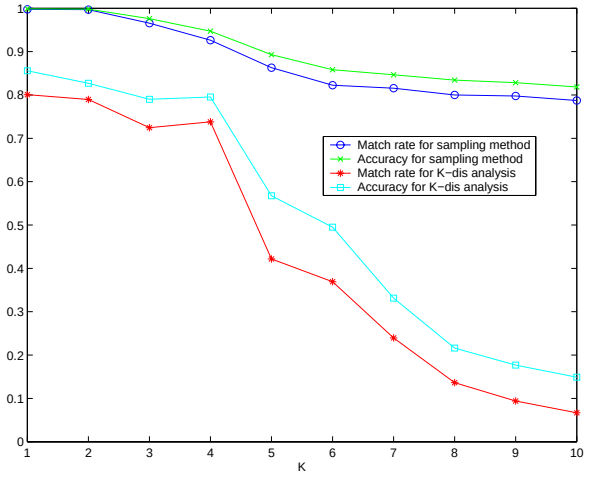


Figure 11: Effect of k on affinity groups with various frequencies

separate group. When the number of groups increases, k-distance analysis can recognize more correct groups. But when the number of groups becomes even larger, the number of occurrences of each data element becomes smaller, since the total size is fixed. This increases the difficulty of recognizing groups. Figures 15 and 16 compare the scalability using a fixed frequency but varied group sizes. In both cases, the sampling method performs much better than k-distance analysis method. The performance of the sampling method remains the same. However, the performance of k-distance analysis decreases when the number affinity groups increases. This is not surprising because when the number of affinity groups increases, more groups have similar reuse signatures, since every group has roughly 400 occurrences.

5. RELATED WORK

The concept of locality can be used in many application domains. We review previous work in four related areas: compiler analysis, web log analysis, and file systems, sequential pattern mining.

Compiler analysis Studying the relations between data elements started as early as 80s. Thabit [24] measured how often a pair of data elements were used together. Chilimbi [2] used grammar compression to find repeated sequence called *hot data streams*. The previous studies focused on finding repeating sequences by measuring frequency and individual similarity. Ding and Zhong [6] proposed to use reuse distance analysis to predict the behavior of repeated data elements. The concept of *reference affinity* was first introduced for trace-level compiler analysis by Zhong and Ding *et al.*in [29]. A formal model of strict reference affinity was proved to guarantee a hierarchical partition of the program data. They also proposed a k-distance analysis approach based on the reuse signature to find strict reference affinity groups. Using array regrouping and structure splitting, they achieved great performance improvement on the benchmarks.

Various statistical sampling methods was used to simulate the behaviors of programs using part of the trace. In [5], a sampling method was used for trace simulation to reduce state loss, which was called *state-reduction* method. Lafage and Seznec [15] proposed a sampling method to select a few program execution slices representative of the entire trace for microarchitecture simulations.

Web system As the number of World Wide Web users increases at a rapid rate in the last decade, web server performance and network traffic became a popular problem. Two popular techniques were used to relieve this situation: prefetching and caching. In most of previous studies decision of prefetching was based on the static hyperlink relationship [3] or simple frequency of hyperlinks [7]. Web logs were used to get the repetitive subsequence of accesses of files in [21]. N-gram model and association rules were also used to find the relationship between files from web logs in [23; 25; 26].

File system In [30], Zhou *et al.*used temporal distance histogram to analyze the access frequency for blocks. Li *et al.*proposed to apply frequent sequence mining method on trace of blocks to find block correlations in [16]. Jiang and Zhang used reuse distance to analyze the access patterns for file caching in [13; 14]. The repeated patterns or the frequency information was then used to direct the cache replacement, block prefetching, and block layout.

Frequent sequence mining The concept of sequential pattern mining was first introduced by Agrwal and Srikant in [1]. Huge amount of work has been done both on efficient sequential pattern mining algorithms [1; 4; 9; 10; 12; 18; 20] and applications [2; 16; 17; 21]. Affinity is similar to the frequent sequence mining. They all try to find the relationship between data elements from the whole trace. But they are different. In frequent sequence mining, we want to find an ordered, repetitive or nearly repetitive subsequence that is above some threshold. But in affinity group analysis, we don't care about the order of data accesses. The concept of

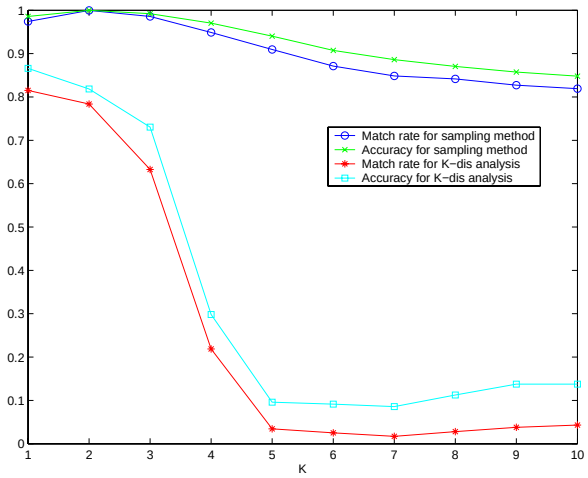


Figure 12: Effect of k on affinity groups with various length

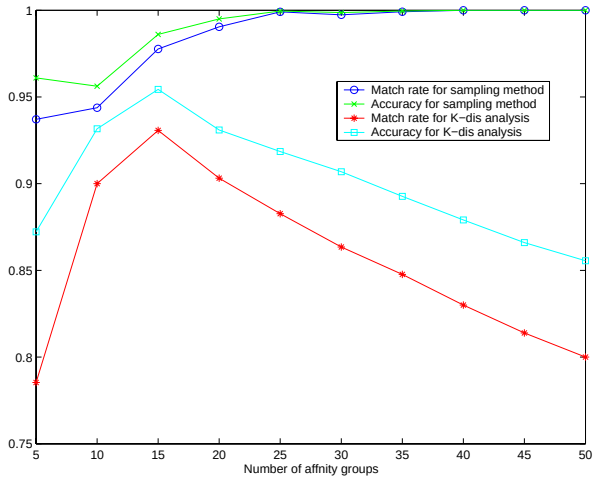


Figure 13: Comparison on linear frequent strict affinity groups

reuse distance allows affinity groups to appear in different orders and different affinity groups overlap in a trace.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced the reference affinity as a model for data locality. We extended strict reference affinity to weak reference affinity. Similar to strict reference affinity, weak reference affinity can also give a unique partition of data elements. Different weak reference affinity groups corresponding to the reuse distance k and affinity threshold θ forms a lattice structure. We also proposed a sampling method for both the weak reference affinity problem and strict reference affinity problem. Experimental results showed that the sampling method outperforms k-distance analysis in terms of scalability and accuracy on reference affinity groups of different weakness, reuse distance k, length and frequency.

For future work, we want to test the sampling algorithm on real applications. Specifically, we want to collect traces from programs, file systems and web systems and then evaluate how well our algorithm works on them. We'll also investigate ways of making use of the reference affinity.

7. REFERENCES

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 3–14. IEEE Computer Society, 1995.

[2] T. M. Chilimbi. Efficient representations and abstractions for quantifying and exploiting data reference locality. In *Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation*, pages 191–202. ACM Press, 2001.

[3] K. Chinen and S. Yamaguchi. An interactive prefetching proxy server for improvements of WWW latency. In *Proceedings of seventh annual conference of the Internet society*, Kuala Lumpur, Malaysia, June 1997.

[4] D. Chudova and P. Smyth. Pattern discovery in sequences under a Markov assumption. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 153–162. ACM Press, 2002.

[5] T. M. Conte, M. A. Hirsch, and K. N. Menezes. Reducing state loss for effective trace sampling of superscalar processors. In *Proceedings of the 1996 International Conference on Computer Design, VLSI in Computers and Processors*, pages 468–477. IEEE Computer Society, 1996.

[6] C. Ding and Y. Zhong. Predicting whole-program locality with reuse distance analysis. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, San Diego, CA, June 2003.

[7] D. Duchamp. Prefetching hyperlinks. In *USENIX Symposium on Internet Technologies and Systems (USITS '99)*, Boulder, CO, 1999.

[8] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.

[9] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. FreeSpan: frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 355–359. ACM Press, 2000.

[10] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 1–12. ACM Press, 2000.

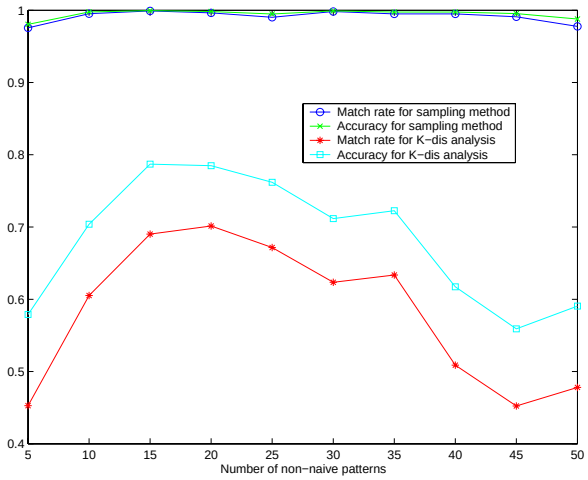


Figure 14: Comparison on linear frequent weak affinity groups

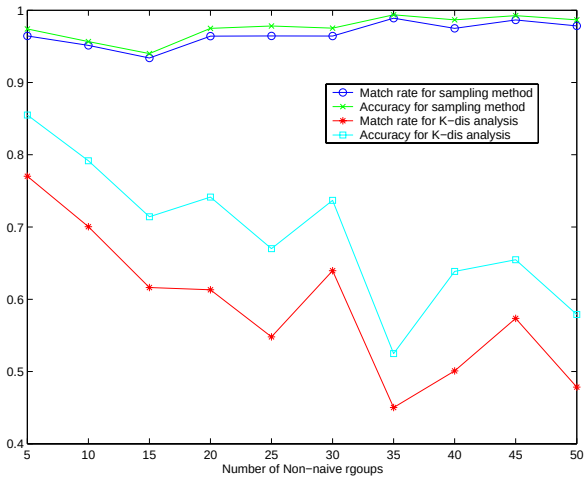


Figure 15: Comparison on linear sized strict affinity groups

[11] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufman, CA, 1996.

[12] M. Hirao, H. Hoshino, A. Shinohara, M. Takeda, and S. Arikawa. A practical algorithm to find the best subsequence patterns. *Theoretical Computer Science*, 292(2):465–479, 2003.

[13] S. Jiang and X. Zhang. LIRS: An efficient low interreference recency set replacement policy to improve buffer cache performance. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 31–42, Marina Del Rey, CA, June 2002.

[14] S. Jiang and X. Zhang. ULC: A file block placement and replacement protocol to effectively exploit hierarchical locality in multi-level buffer caches. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS 2004)*, Tokyo, Japan, March 2004.

[15] T. Lafage and A. Seznec. Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream. In *Proceedings of the Third IEEE Annual Workshop on Workload Characterization*, pages 102–110, September 2000.

[16] Z. Li, Z. Chen, S. Srinivasan, and Y. Zhou. C-Miner: Mining block correlations in storage systems. In *Proceedings of the 3rd USENIX Conference on File and Storage Technology (FAST'04)*, San Francisco,CA, March 2004.

[17] H. Lu, J. Han, and L. Feng. Stock movement and N-dimensional inter-transaction association rules. In *Proceedings of 1998 SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery DMKD'98*, Seattle, Washington, June 1998.

[18] 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.

[19] R. L. Mattson, J. Gecsei, D. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM System Journal*, 9(2):78–117, 1970.

[20] J. Pei, J. Han, and W. Wang. Mining sequential patterns with constraints in large databases. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 18–25. ACM Press, 2002.

[21] J. Pitkow and P. Pirolli. Mining longest repeating subsequence to predict world wide web surfing. In *2nd USENIX Symposium on Internet Technologies and Systems*, Boulder, CO, 1999.

[22] T. Scheffer and S. Wrobel. A sequential sampling algorithm for a general class of utility criteria. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 330–334. ACM Press, 2000.

[23] Z. Su, Q. Yang, Y. Lu, and H. Zhang. WhatNext: A prediction system for web requests using N-gram sequence models. In *Proceedings of the First International Conference on Web Information Systems Engineering (WISE'00)-Volume 1*, page 214. IEEE Computer Society, 2000.

[24] K. O. Thabit. *Cache management by the compiler*. PhD thesis, Department of Computer Science, Rice university, 1981.

[25] Q. Yang and H. H. Zhang. Integrating web prefetching and caching using prediction models. *World Wide Web*, 4(4):299–321, 2001.

[26] Q. Yang, H. H. Zhang, and T. Li. Mining web logs for prediction models in WWW caching and prefetching. In

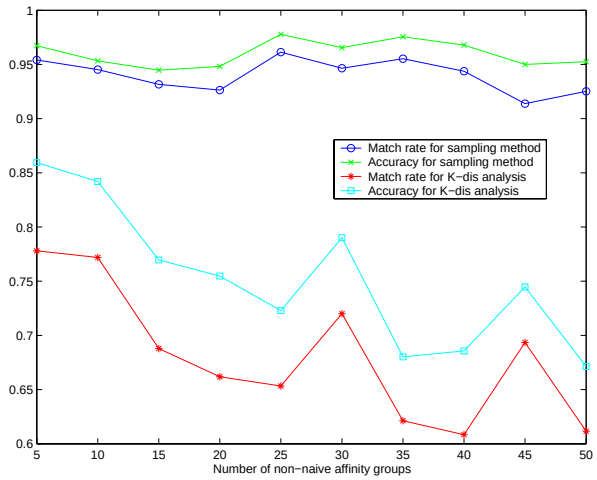


Figure 16: Comparison on linear sized the weak affinity groups

Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, pages 473–478. ACM Press, 2001.

[27] M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, 2001.

[28] C. Zhang, Y. Zhong, C. Ding, and M. Ogihara. A method for hierarchical data placement. Technical Report TR 845, Department of Computer Science, University of Rochester, Aug 2004.

[29] Y. Zhong, M. Orlovich, X. Shen, and C. Ding. Array regrouping and structure splitting using whole-program reference affinity. In *Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation*, pages 255–266. ACM Press, 2004.

[30] Y. Zhou, J. Philbin, and K. Li. The Multi-Queue replacement algorithm for second level buffer caches. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 91–104. USENIX Association, 2001.

Frequent Pattern Mining in Real-Time – First Results

Rajanish Dass
Indian Institute of Management Calcutta
email: rajanish@iimcal.ac.in

Ambuj Mahanti
Indian Institute of Management Calcutta
email: am@iimcal.ac.in

Contact Information

Rajanish Dass
Doctoral Candidate
Fellow Programme: Management Information Systems
Indian Institute of Management Calcutta
Joka
D.H.Road
Kolkata – 700 104.
INDIA

Contact Numbers:
+91-33-24678300-04 Extn: 298 (Office)
+91-33-24726181 (residence)
+91-9830175118 (mobile)

e-mail: rajanish@iimcal.ac.in
rajanishdass@yahoo.com

Ambuj Mahanti
Professor
Management Information Systems
Indian Institute of Management Calcutta
Joka
D.H.Road
Kolkata – 700 104
INDIA

Contact Information:
+91-33-24678300-04 Extn: 405 (Office)
+91-33-24749657 (Residence)

email: am@iimcal.ac.in

Frequent Pattern Mining in Real-Time – First Results

Rajanish Dass
Indian Institute of Management Calcutta
email: rajanish@iimcal.ac.in

Ambuj Mahanti
Indian Institute of Management Calcutta
email: am@iimcal.ac.in

ABSTRACT

Finding frequent patterns from databases has been the most time consuming process of the association rule mining. Till date, a large number of algorithms have been proposed in the area of frequent pattern generation. However, all of these algorithms produce output only at the completion and are not amenable to the real-time need. The need for real-time frequent pattern mining for online tasks and real-time decision-making is increasingly being felt. In this paper, we describe BDFS(b), an algorithm to perform real-time frequent pattern mining using limited computer memory. Empirical evaluations show that our algorithm can make a fair estimation of the probable frequent patterns and reaches some of the longest frequent patterns much faster than the existing algorithms.

1. INTRODUCTION

Since its inception in 1993 by Agarwal et al., association rule mining for large databases of business data, such as transaction records, is of great interest in data mining and knowledge discovery [1]. An association rule is an expression of the form $X \Rightarrow Y$, where *X* and *Y* are sets of items. Such a rule reveals that the transactions in the database, containing items in *X* tend to contain items in *Y*, and the probability, measured as the fractions of the transactions containing *X* also containing *Y*, is called the confidence of the rule. The support of the rule, is the fraction of the total transactions that contain all items both in *X* and *Y*.

For an association rule to hold, the support and the confidence of the rule should satisfy a *user-specified* minimum support and minimum confidence. The problem of mining association rules is to discover all rules that satisfy this user-specified minimum support and minimum confidence. In this paper, we assume that the reader knows the basic assumptions and terminologies of association mining.

However, it is noteworthy at this point that the work of association rule mining can be decomposed into two phases:

1. Frequent itemsets generation: Find out all itemsets (or group of parameters) that exceed the given minimum support
2. Rules construction: From the frequent itemsets generated in step 1 above, generate all association rules having confidence higher than the given minimum confidence.

As the second phase mentioned above is straightforward and less-expensive, researchers have generally focused on the first phase itself. The search space needed for finding all frequent itemsets is undoubtedly huge [4]. A number of efficient algorithms have been proposed in the last few years to make this search fast and accurate[5]. However, most of the algorithms stop only after finding the exhaustive (optimal) set of frequent itemsets. These algorithms have been very efficient and scalable for many real-life applications and are based on the "collect-store-analyze" model. In all these, data mining is typically considered to be an offline analytical task. These algorithms do not have the ability to run under user defined real-time constraints and produce some satisficing (interesting sub-optimal) solutions. With the increasing demand of real-time applications in various fields of business today, development of real-time data mining algorithms demand attention.

In this paper, we describe BDFS(b) (adopted from [16]), a real-time frequent pattern mining algorithm which runs under limited computer memory. We also show its edge over existing efficient association mining algorithms such as FP tree, when it runs to completion and outputs exhaustive set of frequent patterns.

The rest of the paper is organized as follows. In the next section we discuss on the importance of real-time frequent pattern mining in businesses. In Section 3, we present a review of the previous work in association rule mining. In Section 4, we introduce algorithm BDFS(b). Section 5 contains the empirical evaluation of our algorithm. Finally, we conclude the paper in Section 6.

2. NEED FOR REAL-TIME FREQUENT PATTERN MINING IN BUSINESS

In recent years, business intelligence systems are playing pivotal roles in fine-tuning business goals such as improving customer retention, market penetration, profitability and efficiency. In most cases, these insights are driven by analyses of historic data. Now the issue is, if the historic data can help us make better decisions, how real-time data can improve the decision making process [6] ?

An offline approach to data mining reflects sound practice because the data have to be cleaned, checked for accuracy, etc. However, in a scenario of cutthroat competition, the organizations cannot afford to show the attitude of not keeping abreast with the latest changing demands and trends of their customers and get satisfied with periodical data. They have to act on the latest data that is available to them to react not only to the fierce global competition, but also market products keeping in mind of the latest customer wishes. In such a scenario, the concept of a real-time enterprise has creped into the corporate boardrooms of a number of organizations. Using up-to-date information, getting rid of delays, and using speed for competitive advantage is what the real-time enterprise is about [3].

There are numerous areas where real-time decision making plays a crucial role. These include areas like real-time CRM and recommender systems [14, 20], real-time supply chain management [11], real-time enterprise risk and vulnerability management [17], real-time stock management and vendor inventory [21], real-time operational management with special applications in mission critical real-time information as is used in the airlines industry, real-time intrusion and real-time fraud detection [13], real-time negotiations and other areas like real-time dynamic pricing and discount offering to customers in real-time. More than that, real-time data mining will have tremendous importance in areas where a real-time decision can make the difference between life and death – mining patterns in medical systems.

3. PREVIOUS WORK

Association rule mining was introduced by Agrawal et al. [1]. A detailed discussion about the various algorithms of frequent pattern mining and their performance can be found in the literature surveys of frequent pattern mining [5, 7, 10].

It is noteworthy at this point that the total search space for all frequent itemsets is huge. Instead of generating and counting the supports of all possible itemsets at once, which is obviously infeasible, several solutions have been proposed to perform a more directed search by iteratively generating and counting sets of *candidate* itemsets [4].

The most well known and influential algorithms are Apriori[1] and FP-growth [8]. Apriori uses an a-priori knowledge of frequent *k*-itemsets to generate candidate itemsets of length *(k+1)* and employs an innovative technique for pruning non-promising candidates. However, the most discussed drawback of this algorithm is that when the cardinality of the longest frequent itemsets is *k*, Apriori needs *k* passes of database scans. FP-growth, however, uses a depth-first strategy for finding frequent patterns without generating any candidate itemset. It constructs an FP-tree with itemsets above the user-given support and then recursively mines the constructed Fp-tree to find out all patterns. FP-growth makes only 2 scans of the database for finding all the frequent patterns. Many variants of Apriori algorithm have been designed. Other ways of solving the problem were using various partitioning methods and sampling methods. These implementations included algorithms like Apriori-TID[2], Apriori-Hybrid[2], Partition [19], Sampling [23], CARMA [9], ECLAT [15], and Top-Down [15]among others.

Majority of the algorithms in this area have been classified according to their strategy to traverse the search space and by their strategy to determine the support values of the itemsets [10]. However, [22] has concluded that the most salient features of these algorithms are their *counting strategy*, *search direction* and *search strategy*. Horizontal counting or vertical intersections are used for counting the occurrences of candidate itemsets. Most of the algorithms have generally used a bottom up approach in the search strategy. While applying the search strategy, the algorithms have used a breadth first or a depth first search. The above points may be summarized in the following table:

Counting Strategy	**Search Direction**			
	Bottom-up		*Top-Down*	
	Search Strategy		**Search Strategy**	
	Depth-first	*Breadth-first*	*Depth-first*	*Breadth-first*
Counting	FP-Growth	Apriori		Top-Down
Intersection	ECLAT	Partition		

Figure 1. Classification of prevailing algorithms *[22]*

4. BDFS(b): AN EFFICIENT TECHNIQUE OF FREQUENT PATTERN MINING IN REAL-TIME

4.1 Algorithm Basics

In this study, we propose a brute force algorithm, which is a variant of the Block Depth First Search[16]. We call the algorithm as BDFS(b). BDFS(b) explores the given search space in stages. The search is conducted in a depth first manner, which ensures that patterns of greater length will be preferred over those of comparatively shorter lengths.

We assume that a lower triangular frequency matrix M is created in a pre-processing step, which stores the support independent frequencies of all 1-length and 2-length patterns. Once the user specifies a desired support value, all frequent patterns of length 1 and 2 (meaning F(1) and F(2), where F(n) means frequent pattern of length-n) are obtained from M. Then BDFS(b) starts its search for frequent patterns of higher lengths from this point forward.

The most salient features of BDFS(b) are:

(a) It conducts search in stages and uses back-tracking strategy to run to completion and ensure optimal solution.

(b) It takes a block of candidate patterns b from a global pool, conducts the search by checking the frequency of these patterns in the database. It generates the possible candidate patterns (explained later with an example) of the next higher length from the currently known frequent patterns. These candidate patterns are continued to be explored in a systematic manner until all frequent patterns are generated. The value of the block b is defined by the user using her knowledge and experience (later in the paper, we have shown how the performance of BDFS(b) is affected with changing block size b).

A possible state space diagram of BDFS(b) is shown in figure 2.

The initial state (or the root node) in the state-space is denoted by S_0, which contains the complete set of 2-length frequent patterns F(2). In $S_{0,}$, the set of all candidate patterns of length 3 or more are set to ϕ. In general, by the expansion of a node (which is a block of candidate patterns in this case) we mean:

i. Counting the support frequency of all candidate patterns in the state from the database.
ii. Generating the candidate patterns or patterns of border set of next higher level (explained later in the algorithm and its working through example).
iii. Arranging the candidate patterns according to their merits (explained later) and group them into blocks containing b-patterns each. If the block has empty space, it gets candidate patterns from the previous level. This can be handled using a global pool of candidate patterns that has been sorted in descending order of length. We resolve ties arbitrarily.

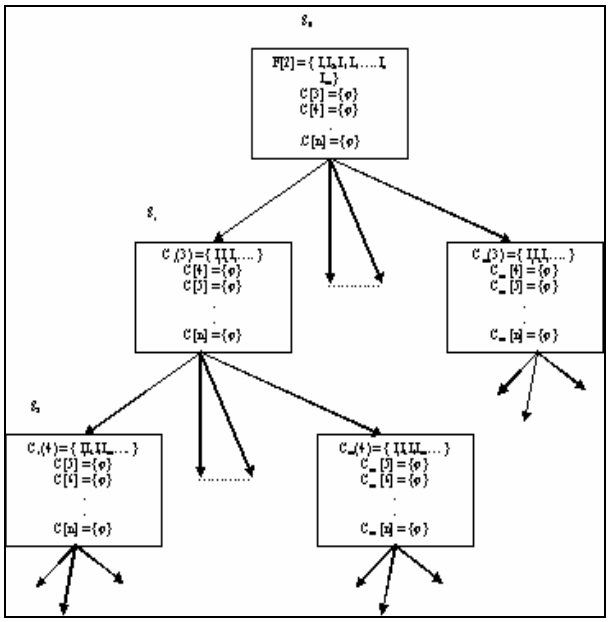


Figure 2. State space diagram for BDFS(b).

4.2 Algorithm Details

Algorithm BDFS(b):

Initialize the allowable execution time τ.
Let the initial search frontier contain all 3-length candidate patterns. Let this search frontier be stored as a global pool of candidate patterns. Initialize a set called Border Set to null.
Order the candidate patterns of the global pool according to their decreasing length (resolve ties arbitrarily). Take a group of most promising candidate patterns and put them in a block b of predefined size.

- *Expand (b)*

Expand (b: block of candidate patterns)
If not last_level
then
begin
Expand$_1$(b)
end.

Expand$_1$(b):

1. *Count support for each candidate pattern in the block b by intersecting the t-id list of the items in the database.*
2. *When a pattern becomes frequent, remove it from the block b and put it in the list of frequent patterns along with its support value. If the pattern is present in the Border Set increase its subitemset counter. If the subitemset counter of the pattern in Border Set is equal to its length move it to the global pool of candidate patterns.*
3. *Prune all patterns whose support values < given*

minimum support. Remove all supersets of these patterns from Border Set.

4. *Generate all patterns of next higher length from the newly obtained frequent patterns at step 3. If all immediate subsets of the newly generated pattern are frequent then put the pattern in the global pool of candidate patterns else put it in the Border Set if the pattern length is > 3.*
5. *Take a block of most promising b candidate patterns from the global pool.*
6. *If block b is empty and no more candidate patterns left, output frequent patterns and exit.*
7. *Call Expand(b) if enough time is left in τ to expand a new block of patterns, ,else output frequent patterns and exit.*

Figure 3. Algorithm BDFS(b)

Let us consider the following example to show how BDFS(b) works. Let us consider some market basket data to illustrate its working, which has been a conventional technique in describing many frequent pattern mining algorithms.

Let the following table represent a set of 12 transactions, where the items are represented by a, b, c …

1. a b c d e	2. a c d e	3. a d e	4. b c d e
5. b d e	6. a b d	7. a b d	8. a b c d
9. d e	10. a c d e	11. a b c d e	12. ace

Figure 4. An example of transaction data

Now we proceed as follows:

Step I. Given this set of transactions D, we create a two dimensional lower triangular matrix M using procedure *Create_Matrix* and the transaction id lists.

I. Create a lower triangular adjacency matrix, M, for n-items (Total storage required: n(n+1)/2). M stores the frequencies of 1-at-a-time and 2-at-a-time combinations of all items.*
II. In M, M(i,j) represents the number of occurrences of the item-pair i and j, $\forall$ i = 1,2…n and $\forall$ j = 1,2,3…i and M(i,i) represents the total number of occurrences of item i.

Figure 5. Procedure *Create_Matrix*

The created matrix M is depicted in figure 6. This step of creating the matrix M and the tid-list (figure 7) is a support independent step and we refer this step through out this paper as a *pre-processing step*.

Step II. Let the absolute support ξ (abs) given for running BDFS(b) be 3. This means that we are interested only in patterns, which have frequency greater than or equal to 3. Cells of Matrix M are visited to find F(1) and F(2) [where F(n) is frequent pattern of length n]. Thus we have:

F(1) = { a(9), b(7), c(7), d(11), e(9)} …… (1)

F(2) = { ab(5), ac(6), ad(8), ae(6), bc(4), bd(7), be(4), cd(6), ce(6), de(8)} ……..…….…… (2)

Frequency of each pattern is shown within parentheses. Thus the pattern *e* of F(1) has frequency 9 and *bd* of F(2) has frequency 7.

Step III. Two 2-length patterns are merged if their first elements match.

Thus *newly merged patterns = { abc, abd, abe, acd, ace, ade, bcd, bce, bde, cde }* ……..…… (3)

Step IV. Find if all the subsets of new merged patterns are frequent. For any 3-lemgth newly merged pattern, if all its 2-length subsets are not present, then the pattern is pruned (using the support monotonicity property[18]). Otherwise, if all its 2-length subsets are present the pattern becomes a *candidate-pattern* and it is moved to the *global-pool* of candidate patterns *C( )*. The global-pool of candidate patterns is sorted on length and any tie between two same length patterns is resolved arbitrarily.

C ( ) = { abc, abd, abe, acd, ace, ade, bcd, bce, bde, cde } ……..……... (4)

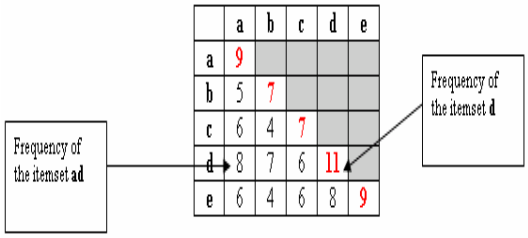


Figure 6. Matrix M

Item	Transaction Ids										
a	1	2	3	6	7	8	10	11	12		
b	1	4	5	6	7	8	11				
c	1	2	4	8	10	11	12				
d	1	2	3	4	5	6	7	8	9	10	11
e	1	2	3	4	5	10	11	12			

Figure 7. The tid-list of the items

Step V. Let us assume that the block size b is 4, which means that we can take at most 4 patterns into a block for checking their frequency. This means as the 3-length candidate patterns are pushed into the global pool, 4 of these patterns namely, *abc, abd, abe* and *acd,* will be put in the next block b.

Step VI. We now check the frequency of these patterns by intersecting the tid-lists of the items.

b = {abc (3), abd (5), abe (2), acd (5)} ..... (5)

As frequency of *abe* is less than the support threshold, it gets pruned.

F (3) = {abc (3), abd (5), acd (5)} ……..…..… (6)

Step VII. We now merge the newly found frequent patterns in *F(3)* and test these newly merged patterns generated for the presence of their immediate subsets..

Newly merged patterns = { abcd } …… (7)

We now find that all immediate subsets of the pattern *abcd* are not present in *F(3)*. But only three immediate subsets are present. Hence we move the pattern *abcd* to border set of length 4, *BS (4*), with a sub-itemset counter of 3.

BS (4) = { abcd (sub-itemset = 3) } ……..…(8)

Patterns *ace, ade, bcd, bce* are taken in the next block b from the global-pool of candidate patterns.

b={ace(5),ade(5),bcd(4),bce(3)}..... (9)

We find that all these items have frequency greater than ξ(abs) = 3 and are hence frequent. Thus from the new block

F(3)={ ace(5) ,ade(5), bcd(4), bce(3)}……..... (10)

For each pattern in the current *F(3)*, we search *BS (4)* to see if any of the immediate supersets are waiting in the border set. We find that the pattern *abcd* is in *BS (4)* with sub-itemset counter = 3. Hence we increase the sub-itemset counter of *abcd* and make it 4. The pattern *abcd* is of the highest length among the candidate patterns in the global-pool and is put in the next block b.

Newly merged patterns (4) = {acde ,bcde } ……(11)

The number of frequent immediate subsets of *acde* and *bcde* are 3 and 2 respectively. Hence they are moved to *BS (4).*

BS (4) = { acde (sub-itemset = 3), bcde (sub-itemset = 2)}…….. (12)

The patterns *abcd, bde* and *cde* go to the current block b. After intersecting the tid-list of these patterns, we find that

F (4) = {abcd (3)} ……..…… (13)
F (3) = {bde (3),cde (5)} …… (14)

Similarly we search the BS (4) with newly found F(3) patterns and merge the patterns in the newly found F(3)'s with previous F(3)'s to generate higher length patterns. We find that *acde* and *bcde* move from BS (4) to global pool of patterns and moves into the block b. By intersecting the tid-lists of the items, we find that

F(4)={acde (4), bcde (3)} ……..…. (15)

As no higher length patterns can be generated and the number of patterns in block b becomes zero and also the number of candidate patterns in the global pool of candidate patterns becomes zero, the algorithm stops executing here. Thus, the set of all frequent patterns are:

F(1) = { a(9), b(7), c(7), d(11), e(9)}

F(2) = { ab(5), ac(6), ad(8), ae(6), bc(4), bd(7), be(4), cd(6), ce(6), de(8)}

F(3) = { abc (3), abd (5), acd (5), ace(5) ,ade(5), bcd(4), bce(3), bde (3),cde (5)}

F(4) = { abcd (3), acde (4), bcde (3)}

The block size b can now be varied to show how it affects the execution time of the algorithm. In the next section, we show and discuss this effect. BDFS(b) has the capability to run in real-time. Whenever it is stopped before its natural completion, it outputs frequent patterns of various lengths it had obtained up to that point of execution time.

5. EMPIRICAL EVALUATIONS

Legend:
T= Average size of transaction; I= Average size of the maximal potentially large itemset; D= No. of transactions in the database; N= Number of items.

In order to show how BDFS(b) performs, when it is run to generate all frequent patterns, we have chosen to compare it with FP-growth and Apriori. Since FP-growth is known to be an order faster and scales better than Apriori[8], we have taken FP-growth as the benchmark and compared its execution time with that of BDFS(b). For the sake of curiosity we have also compared Apriori and BDFS(b) but for their number of patterns checked. The experiments were performed on a Linux machine with 1GB RAM and 20 GB HD.

5.1 Performance of BDFS(b) on Synthetic Datasets

Experimental evaluation of BDFS(b) has been performed on several synthetic datasets like: T10I8D100K,

T10I8D10K, T10I8D1K, T10I2D100K, T6I5D10K, T5I4D1K, T5I4D10K, T5I4D100K (all these datasets have 1K number of items), T5I4N500D1K, T5I4N500D10K, T5I4N500D100K (with number of items being 500), T5I4N100D1K, T5I4N100D10K, T5I4N100D100K (where the number of items is 100) etc. These datasets were generated using the IBM synthetic data generator[1] [2].

5.1.1 Comparison of BDFS(b) with Existing Algorithms

In figures 8 and 9, we have compared the run-time of FP-growth[2] and BDFS(b) for two different datasets and found that BDFS(b) compares well with FP-growth in all the cases. In figure 10, we have tested the scalability of the FP-growth and BDFS(b). We have observed that both the algorithms are well scalable with time and number of transactions in the database. Here again BDFS(b) takes relatively much less time than FP-growth over the same databases.

Comparing the number of patterns being checked by Apriori and BDFS(b), as shown in figure 11, it is found that BDFS(b) checks much lesser number of patterns than Apriori.

5.1.2 Real-time Performance of BDFS(b)

Performance of BDFS(b) for varying values of block size b is shown in figure 12. We find that for b = 1K, 10K and 100K BDFS(b) is well scalable. When the block size is too small, say b = 1K, then it takes more running time for completion compared to b = 10K or 100K. For b = 10K or above it gives similar performance. This is quite intuitive, because a successor block can then accommodate all the candidate patterns of a parent block.

Figures 13 and 14 summarize the real-time behavior of BDFS(b) by depicting the percentage of frequent patterns generated with the percentage of total execution time. These include F(1) and F(2) obtained from the frequency matrix. Figure 14 particularly shows how the percentage of patterns generated by BDFS(b) increases with the increasing values of support for a particular percentage of its running time. Next three figures namely, 15,16 and 17 present three different scenarios of real-time performance of BDFS(b) by showing the number of patterns of different lengths obtained at different time slices (expressed as % of total execution time), using three different block sizes.

[1] The data generator is available from http://www.almaden.ibm.com/cs/quest//syndata.html#assocSynData.

[2] The FP-growth code used for comparison is publicly available at www.cse.cuhk.edu.hk/~kdd/program.html

5.2 Performance of BDFS(b) on Real-life Datasets

To evaluate the performance of BDFS(b), we have tested it on various datasets. This includes real-life datasets like BMS-WebView-1, BMS-WebView-2 and BMS-POS [12]

5.2.1 Comparison of BDFS(b) with Existing Algorithms

In figures 18 and 19, we have compared the run-time of BDFS(b) and FP-growth on two different real-life databases, BMS-POS and BMS-WebView-2. We have found that BDFS(b) performs well on these datasets too.

5.2.2 Real-time Performance of BDFS(b)

In figures 20,21,22,23 and 24, we present the real-time performance of BDFS(b) on three real-life datasets namely, BMS-WebView-2, BMS-POS and BMS-WebView-1 [22]. In figure 20 we show how BDFS(b) performs on the dataset BMS-POS. Again, in figures 21 and 22, we observe the real-time performance of BDFS(b) on real-life datasets BMS-WebView-2 and BMS-WebView-1respectively. In these figures, 20, 21 and 22 we show percentage of frequent patterns generated with percentage execution time having F(1) and F(2) included and excluded in two respective curves. Similarly, in figures 23 and 24, we particularly see that the efficiency of the algorithm enhances with increase in the support value.

Figure 25 makes a tabular presentation of real-time outputs showing length-wise frequent patterns, border sets, and candidate sets, at different time slices (expressed as % of total execution time). It may be seen from the output that all the F(8) patterns (which are of maximal length in this case) were outputted only in 4.17% time. It may be noted that the over all percentage of output is almost always ahead of percentage execution time.

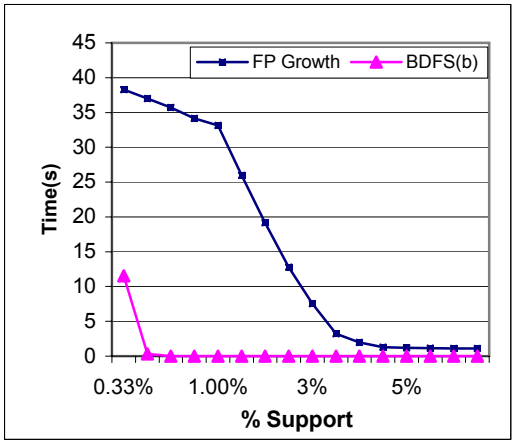


Figure 8. Time Comparison of FP-growth and BDFS(b) T10I8D100K,b =100K

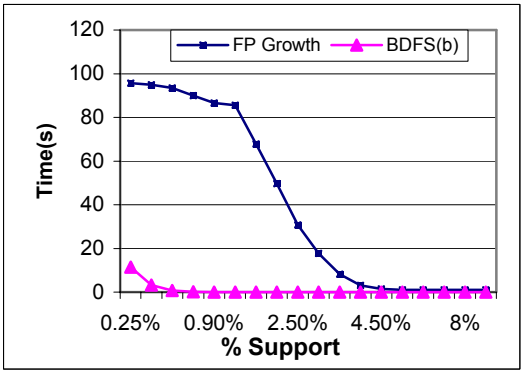


Figure 9. Time Comparison of FP-growth and BDFS(b) T10I2D100K, b=100K

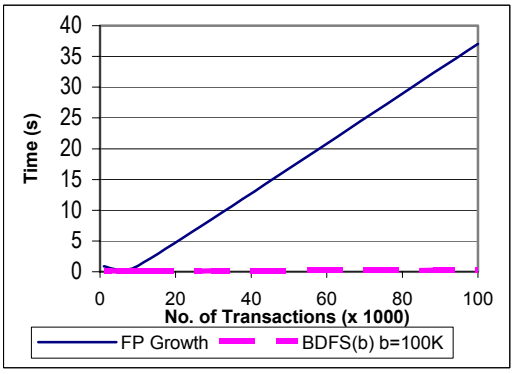


Figure 10. Scalability evaluation of FP-growth and BDFS(b) (b=100K) with varying number of transactions for T10I8 with support=0.5%

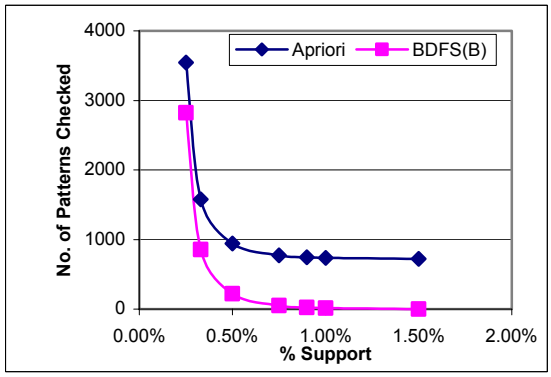


Figure 11. Number of patterns checked by Apriori and BDFS(b) for T10I2D100K with varying supports

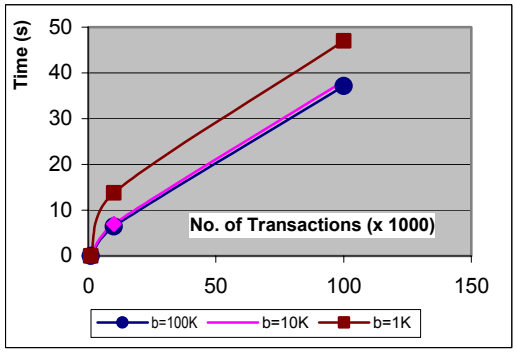


Figure 12. Time Scalability of BDFS(b) with increasing no. of transactions for T5I4D1K,10K,100K and b=1K,10K and 100K and support 0.5%

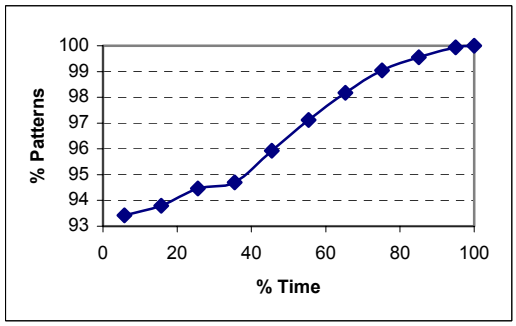


Figure 13. Time-Patterns % for 0.05% support of BDFS(b) for T6I5D10K, BDFS(b), b=1000

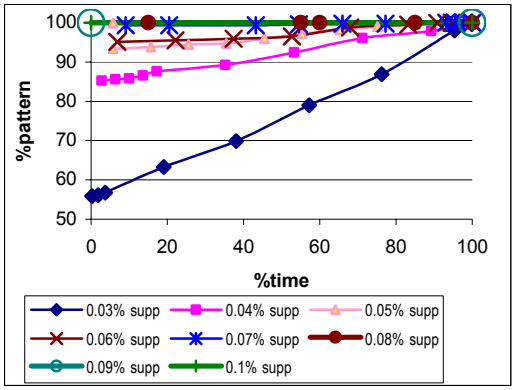


Figure 14. Time-Patterns % for varying support for T6I5D10K, BDFS(b), b=1000

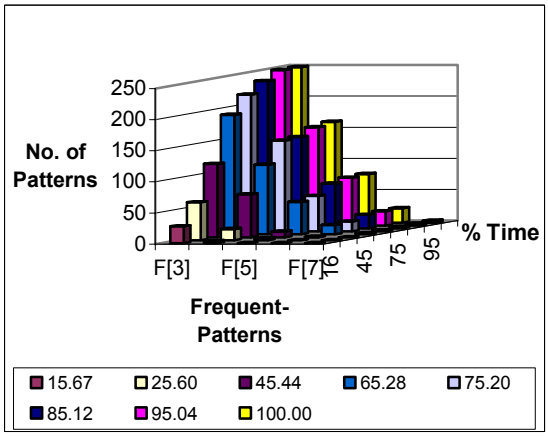


Figure 15. Real-time output of frequent patterns of T6I5D10K, support 0.05%, BDFS(b), b=1000

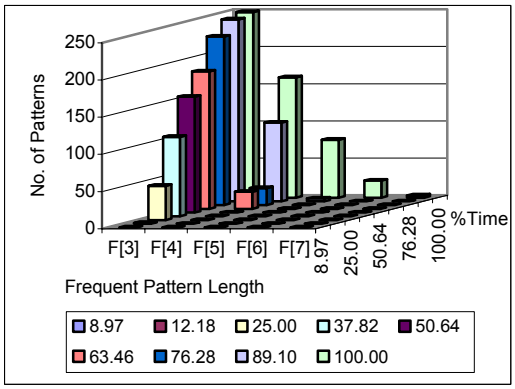


Figure 16. Real-Time output of frequent patterns of T6I5D10K, support 0.05%, BDFS(b), b=10000

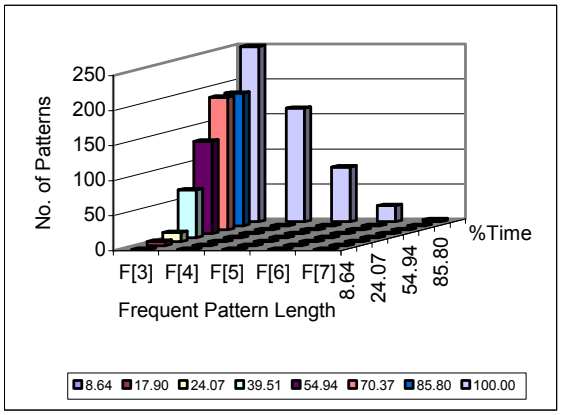


Figure 17. Real-Time output of frequent patterns of T6I5D10K, support 0.05%, BDFS(b), b=100000

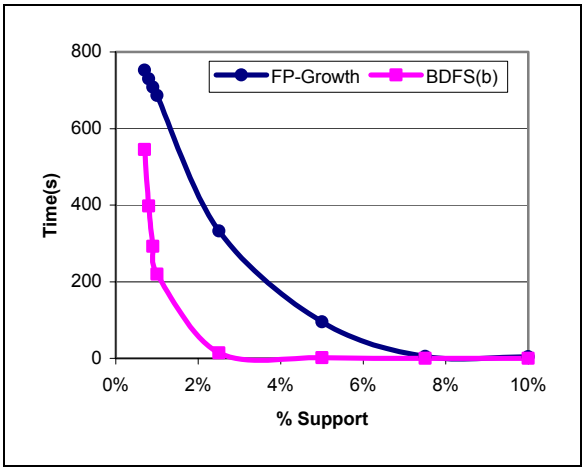


Figure 18. Time Comparison of FP-growth and BDFS(b) for BMS-POS, T=7.5, D = 515,597, N=1658

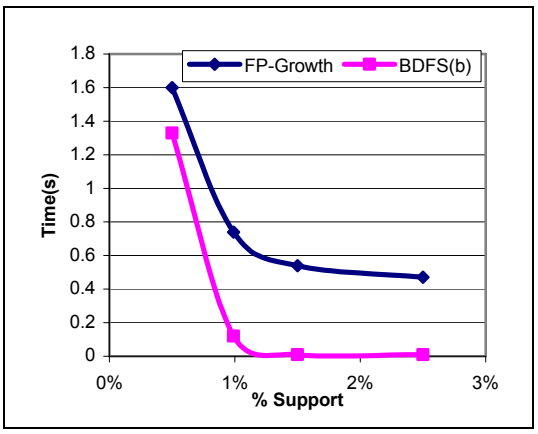


Figure 19. Time Comparison of FP-growth and BDFS(b) for BMS-Web-View-2, T=5.6, D = 77512, N=3341.

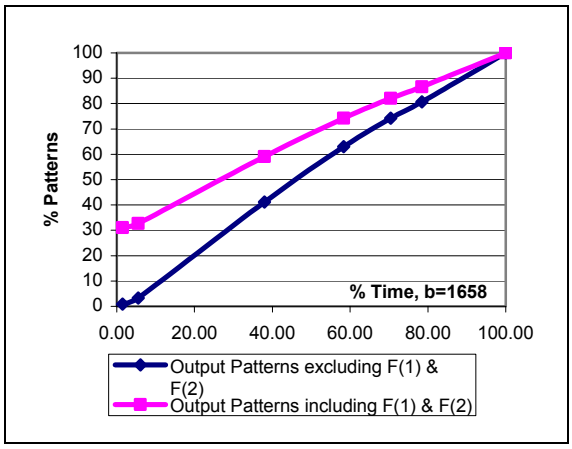


Figure 20. Time-Pattern % for 0.5% support of BDFS(b) for BMS-POS, T=7.5, D = 515,597, N=1658.

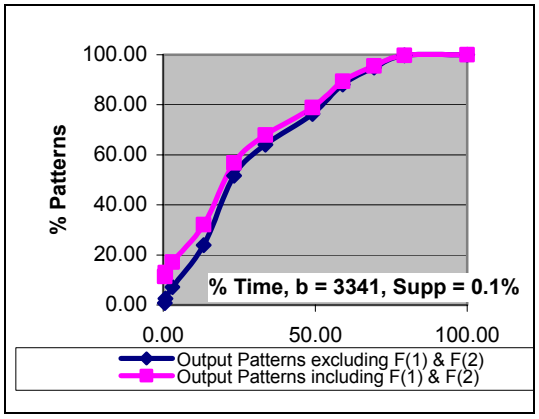


Figure 21. Time-Patterns % for 0.1% support of BDFS(b) for BMS-Web-View-2, T = 5.6, D = 77512, N =3341

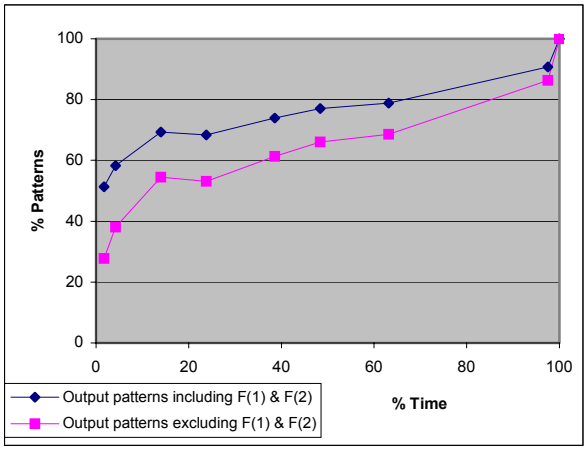


Figure 22. Time-Patterns % of BDFS(b) with b = 497 for BMS-WebView-1, N= 497, T=2.5, D=59602 with support 0.08%

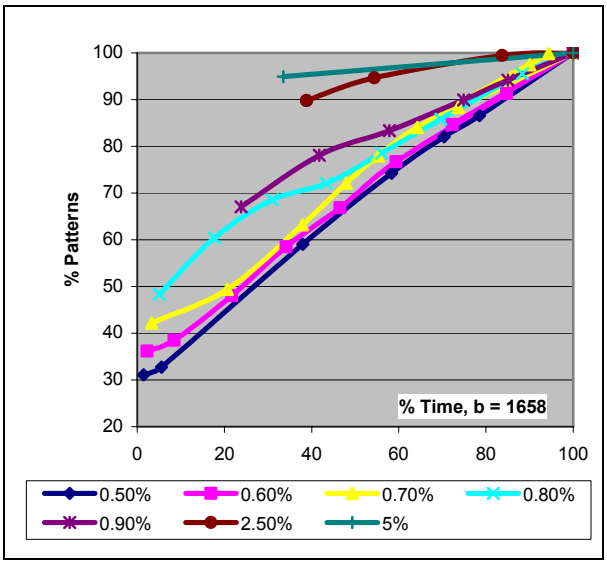


Figure 23. Time-Patterns % of BDFS(b) for varying support for BMS-POS T = 7.5, D = 515597, N = 1658

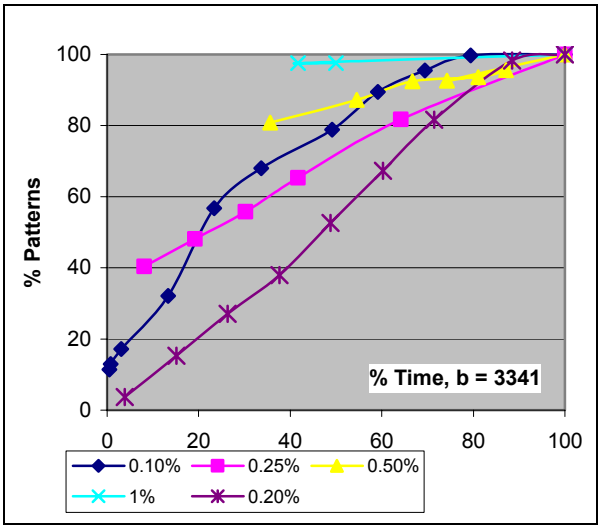


Figure 24.Time-Patterns % of BDFS(b) for varying support for BMS-Web-View-2, T = 5.6, D = 77512, N = 3341

% Time	1.72	4.18	14.01	23.83	38.58	48.41	63.15	97.55	100.00
	F[1] : 350	F[1] : 350	F[1] : 350	F[1] : 350	F[1] : 350	F[1] : 350	F[1] : 350	F[1] : 350	F[1] : 350
	F[2] : 2319	F[2] : 2319	F[2] : 2319	F[2] : 2319	F[2] : 2319	F[2] : 2319	F[2] : 2319	F[2] : 2319	F[2] : 2319
	F[3] : 612	F[3] : 1028	F[3] : 1542	F[3] : 1488	F[3] : 1783	F[3] : 1935	F[3] : 2078	F[3] : 2695	F[3] : 3049
	F[4] : 488	F[4] : 607	F[4] : 953	F[4] : 944	F[4] : 1070	F[4] : 1163	F[4] : 1250	F[4] : 1456	F[4] : 1790
	F[5] : 299	F[5] : 301	F[5] : 347	F[5] : 331	F[5] : 363	F[5] : 376	F[5] : 390	F[5] : 445	F[5] : 508
	F[6] : 134	F[6] : 134	F[6] : 134	F[6] : 134	F[6] : 134	F[6] : 136	F[6] : 136	F[6] : 137	F[6] : 141
	F[7] : 1	F[7] : 31	F[7] : 31	F[7] : 31	F[7] : 31	F[7] : 31	F[7] : 31	F[7] : 31	F[7] : 31
		F[8] : 3	F[8] : 3	F[8] : 3	F[8] : 3	F[8] : 3	F[8] : 3	F[8] : 3	F[8] : 3
Border Sets	BS[4] : 524	BS[4] : 879	BS[4] : 2469	BS[4] : 2446	BS[4] : 2890	BS[4] : 3159	BS[4] : 3676	BS[4] : 4820	BS[4] : 8127
	BS[5] : 285	BS[5] : 310	BS[5] : 625	BS[5] : 625	BS[5] : 697	BS[5] : 747	BS[5] : 771	BS[5] : 927	BS[5] : 1865
	BS[6] : 90	BS[6] : 90	BS[6] : 106	BS[6] : 101	BS[6] : 110	BS[6] : 118	BS[6] : 118	BS[6] : 136	BS[6] : 170
	BS[7] : 21	BS[7] : 21	BS[7] : 21	BS[7] : 21	BS[7] : 21	BS[7] : 21	BS[7] : 21	BS[7] : 21	BS[7] : 21
Candidate Sets	C[3] : 17180	C[3] : 14570	C[3] : 10755	C[3] : 11698	C[3] : 9632	C[3] : 8504	C[3] : 7345		

Figure 25. Frequent Pattern Output along with Border Sets and Candidates Patterns of BDFS(b) for BMS-Web View-1 with support 0.08%. N= 497, T=2.5, D=59602 and b=497.

8. CONCLUSION

Traditionally, the frequent pattern mining has been kept as an offline analytical task, where the frequent patterns are found on the data captured for a specific time period, few weeks, months or even years. But with the changing scenario in the business environment and with improvement in the communications technology and the Internet, and with the more and more business processes going online, frequent pattern mining for real-time decision making has become a thrust area of research [18].

Real-time frequent pattern mining will have great impact on the way knowledge is gathered from patterns from the databases. It has the capability to affect all aspects of doing business in today's world. It will provide decision makers with more accuracy and reduced time lag and help in real-time decision-making.

In this paper, we have proposed an algorithm BDFS(b), which is a brute force version of the Block Depth First Search(BDFS) [16]. First we have compared the performance of BDFS(b) with FP-Growth and Apriori and shown that it does significantly better than both.

Moreover, by adjusting its block size properly, BDFS(b) has the extra ability to run with limited available memory, which often becomes a point of concern in other algorithms. We have then shown that while running under real-time constraints it outputs large chunks of frequent patterns with fractional execution times. We have made detailed performance evaluation based on empirical analysis using several commonly used synthetic datasets and one real-life dataset.

Thus, we have demonstrated that real-time frequent pattern mining can be done successfully using BDFS(b).

Further research in this direction may include design of powerful heuristics to enhance the efficiency of BDFS(b) under different scenarios. We believe this study will encourage use of AI heuristic search techniques in real-time frequent pattern mining.

7. REFERENCE

1. Agarwal, R., Imielinski, T. and Swami, A. Mining Association Rules Between Sets of Items in Large Datasets. in *Proceedings of the ACM SIGMOD Conference on Management of Data*, ACM, Washington,D.C., 1993, 207-216.
2. Agrawal, R. and Srikant, R. Mining Sequential Patterns. in *Proceedings of the 11th IEEE International Conference on Data Engineering*, IEEE, Taipei, Taiwan, 1995.
3. Gartner. The Real-Time Enterprise, 2004.
4. Goethals, B. Memory Issues in Frequent Pattern Mining. in *Proceedings of SAC'04*, ACM, Nicosia, Cyprus, 2004.
5. Goethals, B. Survey on Frequent Pattern Mining, 2003.
6. Gonzales, M.L. Unearth BI in Real-time, Teradata, 2004.
7. Grossman, R.L., Kamath, C., Kegelmeyer, P., Kumar, V. and Namburu, R. *Data Mining for Scientific and Engineering Applications*. Kluwer Academic Publishers, 2001.
8. Han, J., Pei, J. and Yin, Y. Mining Frequent Patterns Without Candidate Generation. in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ACM, Dallas,TX, 2000, 1-12.
9. Hidber, C. Online Association Rule Mining. in *Proceedings ACM SIGMOD International Conference on Management of Data*, ACM, Philadephia, Pennsylvania, 1999, 145-156.
10. Hipp, J., Guntzer, U. and Nakhaeizadeh, G. Algorithms for Association Rule Mining -- A general Survey and Comparision. in *Proceedings of ACM SIGKDD*, ACM, 2000, 58-64.
11. Kalakota, R., Stallaert, J. and Whinston, A.C., Implementing Real time Supply Chain Optimization Systems. in *Supply Chain Management*, (Hong Kong, 1995).
12. Kohavi, R. and Provost, F. Applications of Data Mining to Electronic Commerce. *Journal of Data Mining and Knowledge Discovery*, *5* (2001). 5-10.
13. Lee, W., Stolfo, S.J., Chan, P.K., Eskin, E., Fan, W., Miller, M., Hershkop, S. and Zhang, J., Real time data mining-based intrusion detection. in *DARPA Information Survivability Conference & Exposition II*, (Anaheim, CA , USA, 2001), IEEE Xplore, 89-100 vol. 101.
14. Lin, W. Association Rule Mining for Collaborative Recommender Systems *Computer Science*, Worcester Polytechnique Institute www.wpi.edu/Pubs/ETD/Available/etd-0515100-145926/ unrestricted/wlin.pdf, 2000, 64.
15. M.J.Zaki Scalable Algorithms for Association Mining. *IEEE Transactions on Knowledge and Data Engineering*, *12* (2). 372-390.
16. Mahanti, A., Ghosh, S. and Pal, A.K. A High Performance Limited-Memory Admissible and Real Time Search Algorithm for Networks, University of Maryland at College Park, MD 20742, Maryland, College Park, 1992, 1-15.
17. OpenServiceInc. Real-Time Enterprise Risk and Vulnerability Management, Open Service Incorporation, 2004.
18. Rahman, M.S., Martin, N.L. and Paul, S., Data Mining, Group Memory, Group Decision Making: A Theoretical Framework. in *Ninth Americas Conference on Information Systems*, (2003).
19. Savasere, A., Omiecinski, E. and Navathe, S.B. An Efficient Algorithm for Mining Association Rules in Large Databases. in *Proceedings of the 21nd International Conference on Very Large Databases*, Zurich, Swizerland, 1995, 432-444.
20. Schafer, J., Konstan, J. and Riedl, J. Electronic Commerce Recommender Applications. *Journal of Data Mining and Knowledge Discovery*, *5* (2001). 115-152.
21. SeeBeyond. Real-Time Stock Management and VMI, 2004.
22. Su, J.-H. and Lin, W.Y. CBW: An Efficient Algorithm for Frequent Itmeset Mining. in *Proceedings of the 37th Hawaii International Conference on System Sciences*, IEEE, Hawaii, 2004.
23. Toivonen, H. Sampling Large Databases for Association Rules. in *Proceedings of the 22nd International Conference on Very Large Databases*, Mumbai, India, 1996, 134-145.

A Specialized Binary Tree for Financial Time Series Representation

Tak-chung Fu[1,2], Fu-lai Chung[1,†], Robert Luk[1] and Chak-man Ng[2]

[1]Department of Computing
Hong Kong Polytechnic University
Hunghom, Kowloon, Hong Kong.
{cstcfu, cskchung, csrluk}@comp.polyu.edu.hk

[2]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 – *Effective storage, retrieval and representation of time series are important in many applications. In this paper, a new time series representation scheme that works on the idea of reordering the time series data points according to their importance is proposed. By identifying the extreme points or the so-called perceptually important points of the time series, the importance of each data point can be calculated and a data point importance list can be built. A specialized binary tree (SB-Tree) data structure is proposed to store the importance list. Based on the proposed data structure, a variety of time series analysis and mining tasks can be done with improvement in both efficiency and effectiveness. One may find it particularly attractive in financial time series applications like stock data analysis. Two examples of applying the proposed time series representation scheme are given.*

1. Introduction

Recently, the increasing use of temporal data has initiated various research and development attempts in different fields, e.g., indexing of a set of time series [1], finding similar time series [2] and querying time series database [3]. The problem of how to efficiently and accurately locate patterns (subsequences) of interest in massive time series data is an important and non-trivial problem in a wide variety of applications. While many of the research works have concentrated on this issue, the fundamental problem of how to represent the time series itself has so far not yet been fully addressed. It is important as manipulating a sequence of numbers, possibility a long one, is always difficult.

Recent works in time series representation for data mining tasks can be categorized into two general approaches. The first one transforms the time sequences into the frequency representation while the second one processes the time sequences directly in the time domain. Representations/ Techniques belonging to the first approach include Principal Component Analysis (PCA) [4], Singular Value Decomposition (SVD) [1,5,6], Discrete Fourier Transform (DFT) [7-9], and Discrete Wavelet Transform (DWT) [10-12]. While most of these techniques only keep the first few coefficients to represent the time series, the critical points in the time series are always smoothed out after reconstruction. It is an undesirable behavior particularly in the stock data analysis domain as the locally (and also globally) extreme price values are critical to investment decisions and to characterize the frequently used stock patterns like head-and-shoulder.

On the other hand, the time domain approach works on the temporal data points directly, see e.g., Piecewise Aggregate Approximation (PAA) [1,13], Local Dimensionality Reduction (LDR) [14] and Adaptive Piecewise Constant Approximation (APCA) [5]. Reference [15] introduces a method to search time series data that are first transformed into symbol strings using change ratio between contiguous data points in time series. A suffix tree is then built to index all suffixes of the symbol strings. The focus of this paper is to demonstrate how this kind of methods can adapt to the processing of dynamically constrained time series queries. In [16], a probabilistic model based on linear segmentation of temporal sequence in accordance with prior knowledge for efficient representation is proposed. In [17], an extended representation of time series that allows classification and clustering in addition to the ability to explore time series data in a relevance feedback framework is proposed. It consists of piecewise linear segments to represent shape and a weight vector to contain the relative importance of each individual linear segment. Recently, a symbolic representation of time

† *Corresponding author*

series called Symbolic Aggregate approXimation (SAX) [18] is proposed. It is a time domain approach and has been shown being able to carry out a variety of analysis and mining tasks.

In this paper, a new time series representation scheme for various stock data analysis and mining tasks is proposed. Stock time series has its own characteristics over other time series data (e.g. data from scientific areas like electrocardiogram ECG). For example, it 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 paper is organized into five sections. The proposed representation scheme, which is based on reordering the time series data points according to their importance, is presented in section 2. Section 3 introduces a specialized binary tree (SB-Tree) structure for the proposed representation. It demonstrates how a SB-Tree can built from a given time series and be accessed. In Section 4, two applications of the proposed representation scheme and data structure are reported. The paper is concluded in the final section.

2. 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, different time series retrieval, analysis and mining tasks can be carried out efficiently. 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.

2.1 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 [19] 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 [20] (see also [21]). We also found independent works in [22-24] 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).

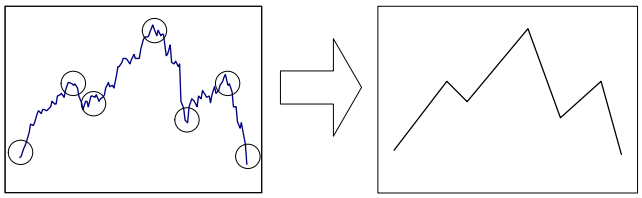


Figure 1. PIP identification results (from 30 data points in the data sequence P to 7 PIPs).

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 Pointlocate (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.

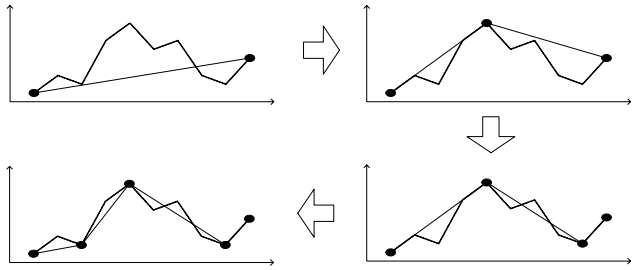


Figure 3. Identification of 5 perceptually important points

2.2 Building Data Point Importance List

Inspired from the above idea, a reordering mechanism for the data points in a time series is proposed here. 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 order 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 I 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).

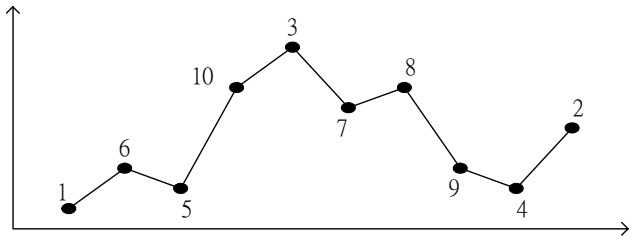


Figure 5 The order of points found by the PIP identification process

Table I. 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. Examples will be shown in section 4.

3. A Specialized Binary Tree (SB-Tree) Data Structure for Time Series Representation

After introducing the concept of the data point importance list for time series representation, we need to tackle an important issue, i.e. how to update the list. As financial time series is usually updated frequently, an algorithm to reduce the time to update the importance list is essential. We here propose to use a binary tree (B-tree) structure to store the data point importance list and describe the mechanisms for building and accessing the tree. We call this structure a specialized binary tree (SB-Tree) and present its operations below. Due to the nature and space limitation of this paper, the updating mechanism is omitted here and will be reported in a coming paper.

3.1 Building SB-Tree

To create a SB-tree, the PIP identification process or the importance list construction process is adopted. The first and last data points in the given sequence are not necessarily added to the tree as they are fixed. The third PIP identified becomes the root node of the tree. Next, a recursive interpretation for the building of a tree is demonstrated. Starting from the root node *ptr*, the current PIP identified forms a *node*:

- If *node->x* is less than *ptr->x* then goto the left arc of *ptr*
  - If *ptr->left* is empty, add *node* to this position
  - Else *ptr = ptr->left* and start the next iteration
- Else goto the right arc of *ptr*
  - If *ptr->right* is empty, add *node* to this position
  - Else *ptr = ptr->right* and start the next iteration

Fig.6 shows the detail algorithm for creating a SB-tree.

```
Function Build_SB_Tree (P)
   Input: sequence P[1..m]
   Output: SBTree root
Begin
   Set list[1]=P[1], list[m]=P[m]
   Repeat until list[1..m] all filled
   Begin
      Select point P[j] with maximum distance dist to
        the adjacent points in list
      Select point P[k] with maximum distance on the
        opposite side
      Append point P[j] to list
      Create node
      node->x=j
      node->y=P[j]
      node->dist=dist
      node->ax=k
      node->ay=P[k]
      node->left=NULL
      node->right=NULL

      If (root = NULL) Then
        root=node
      Else
        ptr=root
        Repeat until all nodes are assigned
        Begin
          If (node->x > ptr->x) Then
            If (ptr->left = NULL) Then
              ptr->left=node
            Else
              ptr=ptr->left
            End
          Else
            If (ptr->right = NULL) Then
              ptr->right=node
            Else
              ptr=ptr->right
            End
          End
        End
      End
   End
   Return root
End
```

Figure 6 Pseudo code of building the SB-Tree

3.2 Converting SB-Tree to Data Point Importance List

Suppose a SB-tree is given and the time series is needed to be retrieved. The SB-tree is accessed recursively, started from the root. In this subsection, the access method of the SB-tree is described and the time series data points will be retrieved according to their importance.

- With respect to the first and the last data points of the time series, the root of the SB-tree is the third PIP. It will be marked as USED when retrieved
- The SB-tree is accessed from the root and each accessible node in each path of the tree is reached. An accessible node is defined as the first node in a path that is not marked as USED
- By comparing the distances among all the accessible nodes, the one with maximum distance is selected as the next PIP. Again, this node will be marked as USED when retrieved
- This process continues until all the nodes in the tree have been processed and retrieved (i.e. marked as USED).

Fig.7 shows the pseudo code for accessing a SB-tree.

```
/* Precondition:
   Set list[1]=P[1], list[m]=P[m]
   Given: sequence P[1..m]
*/
Function Build_List2(root)
   Input: SBTree root
   Output: List list
Begin
   Repeat until list[2..m-1] all filled
   Begin
      node=Find_MAX_PIP(root)
      Append node->x TO list
      Marked node as USED
   End
   Return list
End

Function Find_MAX_PIP
   Input: PIPNode cur_node
   Output: PIPNode Max_node
Begin
   If node NOT marked USED Then
      If (cur_node->dist > Max_node->dist) Then
        Max_node=cur_node
      End
   Else
      If (cur_node->left <> NULL) Then
        Max_node=Find_MAX_PIP(pt->left)
      End
      If (cur->node->right <> NULL) Then
        Max_node=Find_MAX_PIP(pt->right)
      End
   End
   Return Max_node
End
```

Figure 7 Pseudo code of converting SB-Tree to data point importance list

3.3 An Example

In this section, we demonstrate how a SB-tree is built and accessed using the time series in Fig.5 and its importance list in Table I. Based on the given 10 data points, a SB-tree is built and shown in Fig.8. To convert this tree back to its data point importance list, Fig.9 illustrates the access method proposed in section 3.2. Starting from the root node of the

SB-tree, the third PIP (the starting and ending points of the time series are the first two PIPs) is retrieved (i.e. point 5 in Fig.9a) and this node is then marked as USED (filled in black in the diagram). To identify the next PIP, all the accessible nodes in the tree are first identified (circled by bold line) in Fig.9b. Then, the one with maximum distance is identified as the next PIP, i.e. point 9 in Fig.9b. This process continues until all the nodes in the tree are marked, see Fig.9c-9i. The whole time series can then be retrieved in the order of the data point importance in Table I.

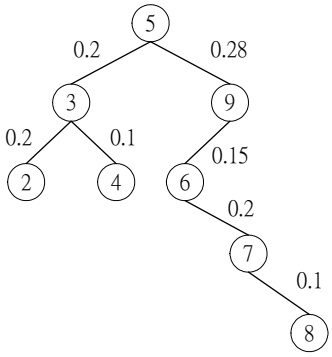


Figure 8. Example of SB-Tree

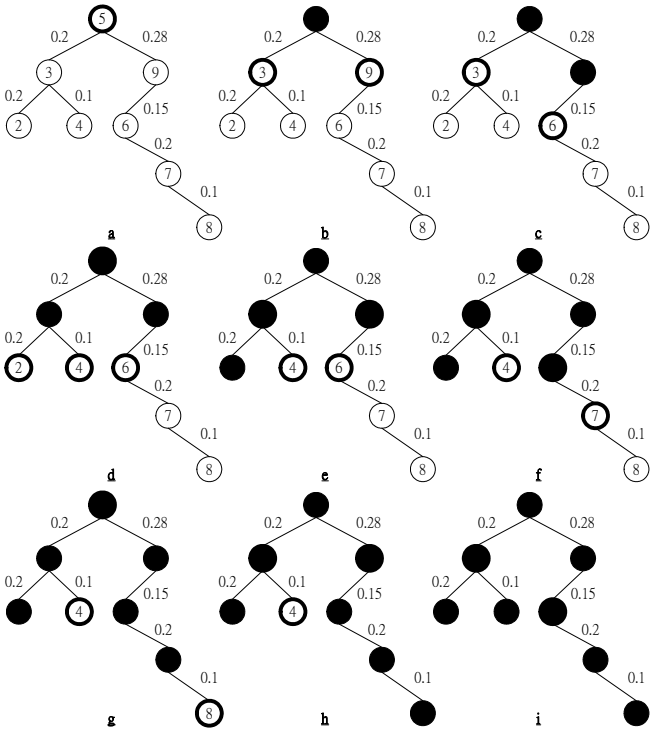


Figure 9. A SB-Tree is accessed to build its data point importance list

4. Potential Applications

As the original time series is transformed and data points are reordered in the new representation, different time series analysis and mining tasks can be considered. Two examples are described in this section. In the first one, the data point importance list of time series is used for multi-resolution visualization. Then, application of the new representation to time series subsequence searching is shown.

4.1 Multi-resolution Visualization

Here, time series with 2500 data points captured from the past 10 years of Hong Kong Hang Seng Index (HSI) in Fig.10 is used. By selecting different number of data points from the data point importance list, different resolutions of the original time series can be shown. It is particularly useful for analyzing time series in different resolutions and transmitting stock price data to mobile devices. In order to display time series data points to either market players or financial analysts for analysis, different resolution is approximated first (e.g. according to the display resolution of the mobile device) and suitable number of data points are then retrieved according to their order in the data point importance list. By doing so, different resolution of the time series can be displayed for different analytical purposes (e.g. long-term vs. short-term). It is especially important for new market analysts to chart with the Wave Principle [25] where noise and unimportant signals should have already been filtered out by the proposed method. The overall picture of the time series can be shown and it is easier to identify a suitable degree of wave and their corresponding type. Examples are given in Fig.11 where the number of data point is selected based on the Fibonacci Sequence).

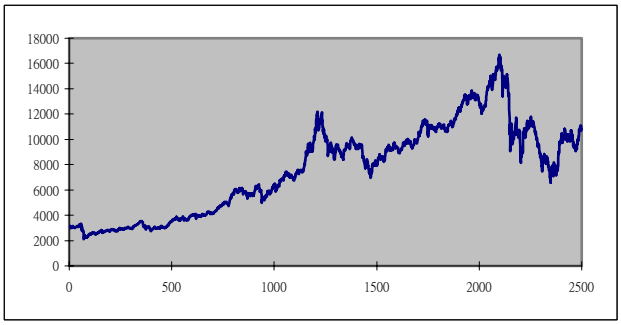


Figure 10. Original Hang Seng index time series with 2500 data points

4.2 Subsequence Search in Time Series

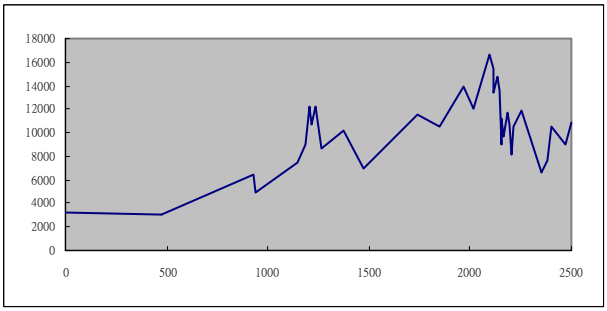
Our second example is subsequence search in time series which can be defined as follows. There are a query sequence Q and a longer time series Y. the task is to find all the subsequences in Y that match Q. Subsequence matching requires the query Q to be placed at every possible offset within the longer sequence Y. To search a specific pattern template within a time series (i.e. subsequence matching), the sliding window approach can be applied. Let l be the number of data points in time series Y and m be the window size. There are l-m+1 subsequences for similarity search and subsequences that fulfill the pattern matching criteria are located.

The similarity between each subsequence $P = (p_1, \cdots, p_m)$ and the query sequence $Q = (q_1, \cdots, q_n)$, where $n \le m$, can be calculated by our previously proposed time series pattern

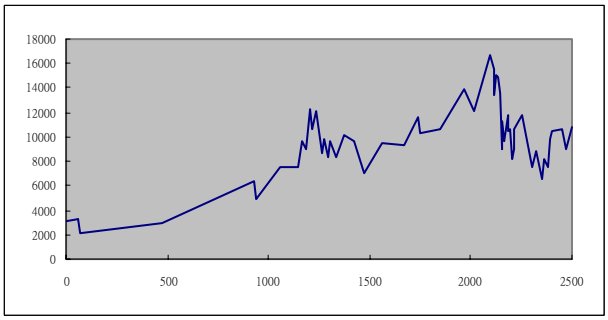
matching scheme [19]. With the proposed time series representation, the same number of subsequences can be searched by scanning from the top of the data point importance list and matching each subsequence formed by the data points with the query pattern. However, an efficient mechanism can be employed here due to the availability of the data point importance list. Except the start and the end points (the first two PIPs) of the subsequence, other data points within that subsequence can be identified based on their order in the data point importance list which is scanned from the top of the list. Furthermore, only the same number of data point as the query sequence, i.e. *n* points, is needed to be extracted from the list.

Fig.12 shows some typical results of identifying a head-and-shoulder (H&S) pattern from different preferable lengths of subsequences (i.e. window size *m*=90, 180 and 360). Fig.13 depicts a speed comparison between the ordinary sliding window approach and the proposed method here. The time needed of the sliding window approach increases linearly due to the need of PIP identification for each window of concern. Thus, longer sequence requires longer time to compute the PIPs. By using the proposed approach, no PIP identification is needed once the importance list is prepared and the speed for subsequence searching can be kept almost constant. Furthermore, the results by both approaches are similar in most cases.

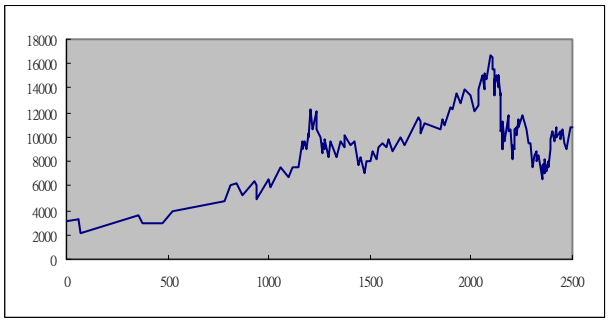
Besides these two applications, the proposed time series representation scheme, characterized by the data point importance list and the SB-tree, can be easily applied to different time series analysis and mining tasks with improvement in both speed and accuracy. One may think of tasks like time series segmentation, indexing, streaming, pattern discovery, and progressive visualization [26].



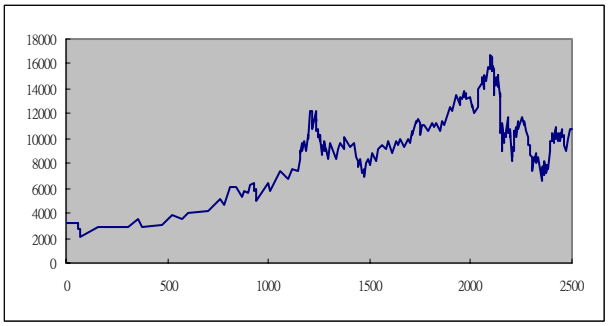
a)



b)

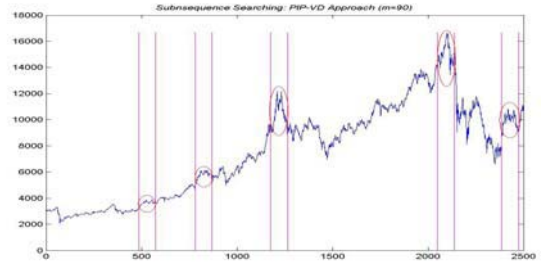


c)



d)

Figure 11. Hang Seng Index time series with 2500 data points in different resolutions: (a) 34, (b) 55, (c) 144 and (d) 233



a)

b)

c)

Figure 12. Identification of "H&S" pattern with different desired length of subsequence (m=90, 180, 360)

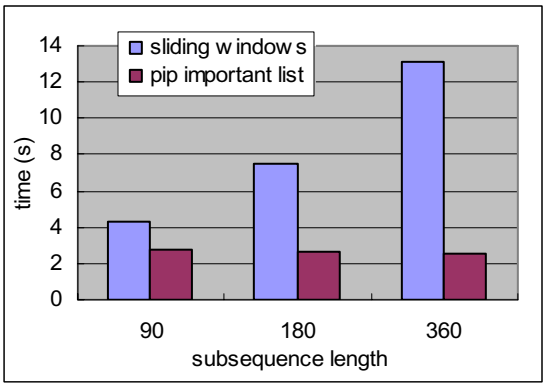


Figure 13. Speed comparison of using sliding window and the proposed approach for subsequence search

5. Conclusions

In this paper, a new time series representation scheme based on reordering the time series data points is proposed. By identifying the extreme points (PIPs) in the time series, the importance of each data point can be determined and a data point importance list can be built. More importantly, an efficient data structure called SB-Tree can be introduced to store the time series data points according to their importance. Such a representation scheme can facilitate a variety of time series retrieval, analysis and mining applications. We have demonstrated the potential of the new representation in multi-resolution time series visualization and time series subsequence search. As already mentioned, new time series data is always available and there is a need to update the importance list or the SB-Tree. We are now working on it and researching some other time series applications and the results will be reported in a coming paper.

References

1. E. Keogh, K. Chakrabarti, M. Pazzani and S. Mehrotra, "Dimensionality reduction for fast similarity search in large time series databases," *Journal of Knowledge and Information Systems*, 2000.
2. G. Das, D. Gunopulos and H. Mannila, "Finding similar time series," *Proc. PKDD'97*, pp.88-95, 1997.
3. R. Agrawal, C. Faloutsos and A.N. Swami, "Efficient similarity search in sequence databases," *Proc. of the 4th Int. Conf. on Foundations of Data Organization and Algorithms*, Chicago, pp.69-84, Oct. 1993.
4. K. Fukunaga, *Introduction to Statistical Pattern Recognition*. Academic Press, second edition, 1990.
5. E. Keogh, K. Chakrabarti, S. Mehrotra and M. Pazzani, "Locally adaptive dimensionality reduction for indexing large time series databases," *Proc. of the 2001 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD 2001)*, pp.151-163, 2001.
6. K. Chakrabarti and S. Mehrotra, "Local dimensionality reduction: A new approach to indexing high dimensional spaces," *Proc. of 26th Int. Conf. on Very Large Data Bases (VLDB 2000)*, pp.89-100, 2000.
7. Y.S. Moon, K.Y. Whang and W.S. Han, "General match: A subsequence matching method in time-series databases based on generalized windows," *Proc. of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 2002)*, pp.382-393, 2002.
8. D. Rafiei and A. Mendelzon, "Querying time series data based on similarity," *IEEE Trans. on Knowledge And Data Engineering*, vol.12, no.5, pp.675-693, 2000.
9. K.K.W. Chu and M.H. Wong, "Fast time-series searching with scaling and shifting," *Proc. of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1999)*, pp.237-248, 1999.
10. I. Popivanov and J. Miller, "Similarity search over time-series data using Wavelets," *Proc. of the 18th Int. Conf. on Data Engineering (ICDE 2002)*, pp.212-224, 2002.
11. T. Kahveci and A. Singh, "Variable length queries for time series data," *Proc. of the 17th Int. Conf. on Data Engineering (ICDE 2001)*, pp.273-282, 2001.
12. K.P. Chan and A.C. Fu, "Efficient time series matching by Wavelets," *Proc. of the 15th International Conference on Data Engineering (ICDE 1999)*, pp.126-133, 1999.
13. B. Yi and C. Faloutsos, "Fast time sequence indexing for arbitrary Lp norms," *Proc. of the 26th Int. Conf. on Very Large Data Bases (VLDB 2000)*, pp.385-394, 2000.
14. E. Keogh and M. Pazzani, "A simple dimensionality reduction technique for fast similarity search in large time series databases," *Proc. of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2000)*, pp.122-133, 2000.
15. Y.W. Huang and P. S. Yu, "Adaptive query processing for time series data," *Proc. of 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, ACM Press, pp.282-286, 15-18 August 1999.
16. E. Keogh and P. Smyth, "A probabilistic approach to fast pattern matching in time series databases," *Proc. of the 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD 1997)*, pp.24-30, 1997.
17. E. Keogh and M. Pazzani, "An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback," *Proc. of the 4th Int. Conf. on Knowledge Discovery and Data Mining*, New York, NY, pp.239-241, 1998.
18. J. Lin, E. Keogh, S. Lonardi and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," *Proc. of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2003.
19. F.L. Chung, T.C. Fu, R. Luk and V. Ng, "Flexible time series pattern matching based on perceptually important points," *Workshop on Learning from Temporal and Spatial Data in Int. Joint Conf. on Artificial Intelligence (IJCAI'01)*, pp.1-7, Seattle, Washington, 4-10 August, 2001.

20. 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.
21. J. Hershberger and J. Snoeyink, “Speeding up the Douglas-Peucker line-simplification algorithm,” *Proc. of the 5th Symp. on Data Handling*, pp.134-143, 1992.
22. C.S. Perng, H. Wang, R. Zhang and D. Parker, “Landmarks: A new model for similarity-based pattern querying in time series databases,” *Proc. of the 16th Int. Conf. on Data Engineering (ICDE 2000)*, pp.33-42, 2000.
23. B. Pratt and E. Fink, “Search for patterns in compressed time series,” *Image and Graphics*, vol.2, no.1, pp.89-106, 2002.
24. E. Fink and B. Pratt, “Indexing of compressed time series,” *Data Mining in Time Series Databases*, pp.51-78, 2003.
25. C.J. Collins, *Elliott Wave Principle*, New Classics Library, Seventh Edition, February 1995.
26. T.C. Fu, F.L. Chung, R. Luk and C.M. Ng, “Progressive time series visualization in a mobile environment,” *The 9th IEEE Symposium on Computers and Communications (ISCC'2004)*, To Appear.

MINING TEMPORAL SEQUENCES TO DISCOVER INTERESTING PATTERNS

Edwin O. Heierman, III, G. Michael Youngblood, Diane J. Cook
Department of Computer Science
The University of Texas at Arlington
Arlington, Texas 76019-0015
{heierman, youngbld, cook@cse.uta.edu

ABSTRACT
When mining temporal sequences, knowledge discovery techniques can be applied that discover interesting patterns of interactions. Existing approaches use frequency, and sometimes length, as measurements for interestingness. Because these are temporal sequences, additional characteristics, such as periodicity, may also be interesting. We propose that information theoretic principles can be used to evaluate interesting characteristics of time-ordered input sequences. In this paper, we present a novel data mining technique based on the Minimum Description Length principle that discovers interesting features in a time-ordered sequence. We discuss features of our real-time mining approach, show applications of the knowledge mined by the approach, and present a technique to bootstrap a decision maker from the mined patterns.

Categories and Subject Descriptors
Mining data streams, novel data mining algorithms, preprocessing and post processing for data mining, spatial and temporal data mining.

General Terms
Algorithms.

Keywords
Discovering interesting episodes, knowledge discovery, mining sequential data streams.

1. INTRODUCTION
With the proliferation of computers comes the proliferation of data created by these computers. Every interaction with a computer system or sensor can be recorded and preserved. It is this abundance of data that has resulted in the emergence of the field known as Data Mining. The number of computer systems and the data these systems collect has far surpassed our ability to review and understand the collected data, so we turn to the computer itself to help us automatically analyze the data for important information.

KDD'04, August 22–25, 2004, Seattle, WA, USA.

Data mining techniques have evolved that perform sequential pattern mining by processing time-ordered input streams and discovering the most frequently occurring patterns. Because these input sequences contain temporal information, additional interesting characteristics, such as periodicity, may also exist in the data. The world, at times, tends to operate in a periodic manner. The earth orbits around the sun every 365 days, and completes a revolution about its axis every twenty-four hours. In addition, mankind has subdivided these periods into years, months, weeks, days, hours, and seconds. By doing so, we can predict when the sun will rise again, when winter will approach, and when the earth once again will orbit around the sun. By taking advantage of the periodic behavior of our environment to organize time, we also tend to exhibit behavior that is periodic. We rise at about the same time everyday to eat. We go to work five days a week, and take the weekends off. Our favorite TV programs are shown at the same time on the same day of each week. These periodic interactions provide predictability and add stability to our lives. Thus, data collected by a computer system responding to interactions with a human or an environmental occurrence may also contain patterns that are governed by periodic behavior.

In our work, we use information theoretic principles to evaluate characteristics in an input sequence other than frequency, such as length and periodicity. By using information theory as a foundation, we anticipate that other characteristics can also be evaluated for interestingness. In this paper, we describe our work, discuss the knowledge that is discovered, and present a technique that uses the mined knowledge to bootstrap a decision maker that operates in a real-time environment.

2. SEQUENTIAL PATTERN MINING
Our work is related to techniques for sequential data mining, which is the task of mining frequently occurring patterns related in time or to other sequences [3]. An example of a sequential pattern is:

> *An individual who bought a car three months ago is likely to change the oil in the car within the next two weeks.*

A common characteristic of techniques that mine sequences is the discovery of patterns that are frequent [1][6]. The more frequently a pattern occurs, the more likely it is that the pattern is important. In addition, some approaches also take into account the length of a pattern [1]. Important knowledge is provided if a

system can report that the symbols *{a, b}* occur frequently together, as opposed to only reporting that *{a}* occurs frequently and *{b}* occurs frequently. However, little attention has been given to evaluating other characteristics of the patterns, such as periodicity. This is surprising, given the emphasis individuals place on organizing their time. Thus, we are interested in a data-mining technique that can discover features about a pattern like the following:

> *At 6:00pm on Monday, Wednesday, and Friday of every other week, Dave leaves work and travels to the gym.*

There are several important attributes of the problem we are investigating that influence our work. First, our data source will be time-ordered input sequences that consist of occurrences of events with no natural points that indicate the start or stop of the pattern. Therefore, our approach must partition the input sequence by grouping together interactions that are related. Second, the nature of the patterns is not known a priori. As a result, the pattern-ordering characteristics (e.g., ordered, unordered) must also be discovered. Third, frequency is only one of the characteristics that should be evaluated. Additional features, such as pattern length and periodicity (occurs daily, weekly), should also be considered. Therefore, the approach will need to balance the values of more than one interestingness measurement.

With these attributes in mind, we have developed an Episode Discovery (ED) algorithm that processes a time-ordered input stream and identifies clusters of events, or episodes, that are closely related in time. These episodes are represented as an unordered set of events. Repeating symbolic patterns in the episodes are evaluated with an approach based on the Minimum Description Length principle. A pattern is described with an encoding, which may result in a reduction in the description length of the original input sequence. The symbolic-pattern encodings that result in the greatest compression of the input sequence are selected as interesting, and the associated patterns are presented to the user as interesting episodes. The details of our approach are presented in the next section.

Several works address mining sequential patterns. Agrawal and Srikant [1] present three techniques for mining sequential patterns from time-ordered transactions. Each transaction is a set of items, and variations of the Apriori property are used to find large sequences by first computing the large itemsets and then constructing the large sequences. Finally, selecting the sequences of largest length first and then pruning the subsets of that sequence yields the set of maximal sequences. Mannila, Toivonen, and Verkamo [6] consider the problem of discovering frequent episodes in an event sequence. A user-defined event window partitions the event stream into overlapping collections of events that are close to each other in time. These collections are examined to find frequent parallel and serial episodes by use of the Apriori property. All frequent episodes are output by the algorithm. Srikant and Agrawal [8] extend their previous work by supporting maximum and minimum time gaps between sequence elements, a sliding time window (event window) across transaction sets, and user-defined taxonomies. The support used for the Apriori-based algorithm is supplanted with additional measures that account for the new features.

3. EPISODE DISCOVERY

We view our data-mining task as the process of describing a time-ordered input sequence with encodings that represent the interesting characteristics that may be present in the sequence. Rissanen's work [7] on the Minimum Description Length (MDL) principle, which proposes searching for models and model classes with the shortest description length, serves as the underlying theory for our work. Rissanen reasons that by using an encoding of a dataset to reduce its description length, constraints are applied to the data that reduce the uncertainty about the nature of the data. Here, encoding is used as a general term to mean an exact representation. The resulting "minimum-encoded" description captures the properties that provide the most likely explanation for the data. Stated another way, the model defines a distribution that assigns the maximum probability to the observed data. Therefore, we will define encodings that encompass interesting characteristics, and use these encodings to find a model that exhibits the minimum description length. The end result will be descriptions that provide a likely explanation of the input.

We reason that if a pattern exhibits periodicity, then describing the pattern with terms that include its periodicity would reduce the overall description length of the input sequence because the occurrences of the periodic pattern can be replaced by the description. In addition, because patterns in the sequence may not be periodic, an encoding will need to be defined that describes non-periodic patterns. Finally, we consider pattern length and frequency to also be interesting characteristics, so the encodings should also describe these features. We anticipate that "more frequent" and "greater length" encodings will also reduce the description length of the input sequence. We now present our MDL-based approach that makes use of encodings to discover if patterns present in a time-ordered input sequence exhibit these interesting characteristics.

3.1 Terminology

We start our discussion by defining the input, processing, and output of our approach. The time-ordered input sequence is defined as follows. We will let Π represent the set of all possible symbols or interactions that could appear in an input stream (the domain alphabet), and let s represent any symbol that is a member of Π. An event δ is denoted as the pair (s, t), where t is a timestamp consisting of time and date information. An event sequence O is defined as an ordered sequence of events, $O = (\delta_1, \delta_2, \ldots, \delta_n)$. The event sequence consists of all events that have occurred up to the point in time represented by t_n. The sequence must be in non-descending order based on the timestamp value, such that $t_i \leq t_{i+1}$. Events in which the timestamp values are the same can be placed in any order.

Our objective is to discover the symbolic patterns in O that are considered interesting. In order to find these patterns, we will collect sets of events from the input sequence into an ordered collection, which we refer to as episodes. Formally, we define an episode ε to be a sequence of event occurrences, $\varepsilon = (\delta_j, \delta_{j+1}, \delta_{j+2}, \ldots, \delta_{j+m})$, which is a sub-sequence of O starting at time stamp

t_j and ending at t_{j+m}. Upon partitioning the input sequence into overlapping episodes, the algorithm will evaluate patterns found in multiple episodes by encoding a description of the pattern occurrences. The encodings that contribute to a minimum description length of the input sequence will be selected as interesting.

ED will evaluate if a pattern occurs at repeatable time intervals. An interval may consist of just a single value or a sequence of values. In order to explain an interval consisting of a single value, assume the existence of symbols *a* and *b*. After partitioning *O* into episodes, ED discovers that the symbols *a* and *b* appear together in multiple episodes, and that these episodes occur every 24 hours. Intuitively, then, this represents a situation in which the symbols *{a, b}* occur on a daily basis.

The interval may also be an ordered sequence of values, such as 48 hours, 48 hours, and 72 hours. Once again using the symbols *a* and *b*, assume ED has discovered a situation in which these symbols appear together in multiple episodes, and that the time interval between the episodes repeats the pattern 48 hours, 48 hours, and 72 hours over and over. We will assume the first episode occurs on Monday. There is 48 hours until the next occurrence of an episode containing *{a, b}*, which now makes it Wednesday. After an additional 48 hours passes, now making it Friday, another episode occurs that contains *{a, b}*. Finally, 72 hours elapses until another occurrence of an episode containing *{a, b}*, which makes it Monday again. Thus, *{a, b}* occurs on Monday, Wednesday, and Friday of every week.

ED will output a collection of interesting episodes, ω, that are discovered in an input sequence *O*. We will let λ, called a symbolic pattern, represent an unordered collection of symbols, $\{s_1, s_2, \ldots, s_n\}$. In the previous discussion, λ would be the set *{a, b}*. The symbolic pattern will be output as part of the discovered knowledge. Because we will treat the pattern as an unordered collection, a set of symbolic patterns that represent the permutations of λ that actually occur in the dataset will be provided as part of the output, represented as $\Lambda=\{\lambda^1, \lambda^2, \ldots, \lambda^m\}$. An example of such a set is $\Lambda=\{(a,b), (b,a)\}$, where $\lambda^1=(a, b)$ and $\lambda^2=(b, a)$. We will let $\mu=(\mu_1, \ldots, \mu_i)$ represent the repeating interval sequence. In the example above, the interval sequence would be $\mu=(48, 48, 72)$. If the interesting episode follows no repeating interval, then the sequence will be empty. Finally, $\Psi=\{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_j\}$ represents the episodes that contain the symbols defined by λ. With this notation, we define an interesting episode as $\omega = \{\lambda, \Lambda, \mu, \Psi\}$.

3.2 Pattern Encodings

We now turn our attention to the encodings that will be used. When encoding a symbolic pattern, we are interested in determining if the episodes containing the pattern repeat according to an interval sequence μ. We will evaluate two types of repeating interval sequences: fined-grained and course-grained. In order to present an example of a fine-grained interval, assume the symbols *{a, b}* occur periodically in multiple episodes on Mondays, Wednesdays, and Fridays. We will assume that *a* always occurs at 12:00:00pm, and that *b* always occurs at 12:00:30pm. An input sequence of events consisting of symbols *a* and *b*, and a possible collection of associated episodes, is shown in Table 1. We will let *x* represent a candidate episode that describes this situation based on the number of hours between the episode occurrences. Upon inspection of the table, it can be seen that *x* has the following characteristics:

$$\lambda_x = \{a, b\},$$
$$\Lambda_x = \{(a, b)\},$$
$$\mu_x = (48, 48, 72), \text{ and}$$
$$\Psi_x = \{\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4, \varepsilon_5, \varepsilon_6, \varepsilon_7\}.$$

In some situations, episodes containing common symbol patterns may not occur at exactly the same exact time interval, but can still be represented by an interval sequence. For example, it may be that an individual washes their clothes every Sunday, but not necessarily at the same time each Sunday. Consider once again the example shown in Table 1. A course-grained interval is computed by partitioning the timestamps into calendar days, which results in $\mu=(2, 2, 3)$. The interval between two timestamps that occur on the same day is zero. The time-based partition could be other categories, such as month or even year. We have chosen to use days because it is a common course-grained interval that balances the granularity of the partitions.

Table 1. Example Input Sequence of *{a, b}*.

Events	Episode	Interval
{*a*, 4/26/04,12:00:00pm} {*b*, 4/26/04,12:00:30pm}	ε_1	
{*a*, 4/28/04,12:00:00pm} {*b*, 4/28/04,12:00:30pm}	ε_2	48 hours 2 days
{*a*, 4/30/04,12:00:00pm} {*b*, 4/30/04,12:00:30pm}	ε_3	48 hours 2 days
{*a*, 5/3/04,12:00:00pm} {*b*, 5/3/04,12:00:30pm}	ε_4	72 hours 3 days
{*a*, 5/5/04,12:00:00pm} {*b*, 5/5/04,12:00:30pm}	ε_5	48 hours 2 days
{*a*, 5/7/04,12:00:00pm} {*b*, 5/7/04, 12:00:30pm}	ε_6	48 hours 2 days
{*a*, 5/10/04, 12:00:00pm} {*b*, 5/10/04, 12:00:30pm}	ε_7	72 hours 3 days

When describing patterns that follow an interval sequence, the encodings must account for mistakes that occur when the episodes do not completely follow the interval sequence. The greater the number of mistakes, the less predictable behavior the pattern displays, and the less the periodic description compresses the original input sequence. Other mistakes, such as a difference in the included events, are not part of the encoding. For example, if the pattern is *{a, b, c}* and the input data contains an occurrence of *{a, b}*, that occurrence of *{a, b}* will be evaluated as a separate pattern. Only if the mistake is a variation in start time is it counted as an encoding mistake.

We have chosen to use three potential encodings for each pattern candidate: one that reflects a fine-grained periodicity, one that reflects a course-grained periodicity, and one that reflects frequency. The periodic encodings include length, frequency, periodicity, and mistakes in periodicity. The encoding for frequency includes frequency and length characteristics, and represents a non-periodic description of the pattern.

Rissanen notes there is no way to determine if an encoding is optimal [7], so we make no guarantee that these encodings are optimal. However, we rely on this premise of the MDL principle:

> *By defining a description and searching for the minimum length using that description, the best model will remain as the best explanation of the observations [7].*

By evaluating encodings and searching for a model that is the minimum description of the original input, at the very least the algorithm discovers a model that can be used for comparison with other models that may be discovered in the future. In addition, we will be selecting the most likely model based on our encoding, which will allow us to determine if patterns do indeed exhibit the characteristics our encodings describe.

Specifically, we will use the three encodings to compute a compression ratio for each candidate pattern. The compression ratio measures how much of the original input sequence is compressed by the encoding. A compression ratio of 2:1 implies that it takes just one unit in the compressed description to represent two units of the original input sequence. Of the three potential encodings, the encoding that results in the greatest compression ratio is selected as the encoding for that repeating pattern. The selected encoding will result in the greatest decrease in the description length of the original input sequence. Once all candidates are assigned an encoding, candidates with the greatest compression ratio will be selected as interesting episodes. By using these encodings, the description length of the input sequence will be reduced. Because candidates that provide the greatest compression are selected, the description with the minimum length will be discovered. If a fine-grained or course-grained encoding is used for an interesting episode, then we consider this candidate to be a *periodic episode.* When the frequency-based encoding is selected, then we use the term *frequent episode* to describe the interesting episode.

Using these encodings, a compression ratio was computed for the input sequence reflected by Table 1 for candidate episode *x*. These values are shown in Table 2. Based on these values, a fine-grained encoding would be selected for *x* because the encoding results in the greatest compression. We would consider *x* to be a periodic episode if it is chosen as an interesting candidate.

Table 2. Summary of Compression Ratios for *x*.

Encoding	Compression Ratio
Fine Grained	1.45
Course Grained	1.20
Frequency	1.27

3.3 Attributes of the Interestingness Measures

ED evaluates the patterns in the input sequence using encodings that incorporate length, frequency, and periodicity. Because of the encodings, the algorithm evaluates the characteristics in the following manner:

- Length - If two patterns have the same frequency and periodicity, the pattern that has a greater pattern length will be more interesting (greater compression ratio).
- Frequency - If two patterns have the same length and periodicity, the pattern that is more frequent will be more interesting.
- Periodicity - If two patterns have the same length and frequency, the pattern that contains fewer periodicity mistakes will be more interesting.

4. ALGORITHM DETAILS

The following summarizes the high-level steps of the algorithm. Given as input a stream *O* of event occurrences δ, ED:

1. Partitions the event sequence O into possibly overlapping maximal episodes, εi, by using an event-folding window W with a time span of tw and a capacity of cw.
2. Creates an initial set of candidate episodes, Ci, from the maximal episodes.
3. Creates additional candidate episodes from the subsets of the maximal episodes.
4. Computes a compression ratio for each Ci.
5. Identifies interesting episodes by evaluating the compression ratios of the candidate episodes. Additional candidate episodes may be generated when a candidate episode is selected as interesting.
6. Outputs a list of interesting episodes.

4.1 Step One: Partition the Input Sequence

ED partitions the input into maximal episodes by incrementally processing the events. An event-folding window collects the events and creates an episode when the time span or capacity of the window exceeds the corresponding parameter value. This is similar to the approach taken by Mannila, et al. [6]. The t_w parameter represents the time span of the window, and c_w the capacity. If $t_w=\infty$, then the window partitions the input sequences into episodes of size c_w. If $c_w=\infty$, then the window partitions the input sequence solely on the time span. The current time interval of the window is calculated based on the time stamp of the event being processed and t_w. Thus, if event δ_i is being added to the window, then the time interval of the window is t_i - t_w. The capacity is computed by counting the number of events currently contained in the window. When one or more events contained in the window are now outside of the specified parameter values due to the addition of the new event, those events are pruned from the window. The window contents prior to pruning are maximal for that particular window instance, and are used to generate a maximal episode. Our approach generates overlapping maximal episodes with potentially different lengths.

Table 3 shows an example of creating maximal episodes with $t_w=15$ and $c_w=\infty$. Five events are incrementally processed. We have simplified this example by considering the timestamp to be an integer value. Because events *(a, 1), (b, 5),* and *(c, 10)* all occur within the fifteen time-unit window, they are accumulated and kept as part of the episode window. When occurrence *(d, 20)* is processed, event *(a, 1)* must be removed from the window. At this point, the episode window contains the maximal episode *((a, 1), (b, 5), (c, 10)).* When event *(e, 40)* is encountered, the

occurrences *(b, 5), (c, 10),* and *(d, 20)* must be removed. Thus, the window contains the maximal episode *((b, 5), (c, 10),* (*d, 20)).* Assuming no other events are processed, the final maximal episode is *((e, 40)).*

Table 3. Creating a Maximal Episode, t_w=15 and c_w=∞.

Event	Episode Window	Start	Stop	Maximal Episodes
(a, 1)	((a, 1))	1	1	
(b, 5)	((a, 1), (b, 5))	1	5	
(c, 10)	((a, 1), (b, 5), (c, 10))	1	10	
(d, 20)	((b, 5), (c, 10), (d, 20))	5	20	((a, 1), (b, 5), (c, 10))
(e, 40)	((e, 40))	40	40	((b, 5), (c, 10), (d, 20))
<end>				((e, 40))

4.2 Steps 2 and 3: Create Candidates

The algorithm constructs the initial collection of candidate episode by creating a corresponding candidate for each maximal episode. However, additional symbolic patterns exist within each maximal episode that may need to be evaluated. For example, if the maximal episode contains the symbolic pattern *{a,b,c,d,e}*, then additional patterns that could be examined would be *{a,b,c,d}*, *{c,d,e}*, and so on. The power set of each symbolic pattern is the complete list of potential patterns that could be evaluated.

One possible approach to generating the additional candidates would be to generate the power set as new candidates. However, this would be intractable because it generates as 2^n candidates, where *n* is the number of symbols in the pattern. Thus, the candidate generation method must prune the complete set of potential candidates in a tractable manner, while ensuring that it does not eliminate any candidates that ultimately do represent interesting episodes.

The Apriori property prunes a search space by deleting non-frequent candidates, and then generating new candidates from the current list of candidates [3]. However, because frequency is not our only discriminator of interestingness, the Apriori approach does not work as a pruning technique. Nevertheless, it is possible to prune the candidate search space by selecting a subset of a symbolic pattern as an additional candidate based on one of the following conditions:

- The subset represents the intersection of a maximal episode with one or more other maximal episodes. Because the subset represents pattern occurrences in multiple episodes, it may be more significant than its parents.
- The subset represents the difference between a maximal episode and one of its episode subsets, which has been selected as an interesting pattern. In this situation, if a subset candidate is evaluated as interesting, then the remainder of the maximal episode must be evaluated to see if it is interesting.

Our approach relies on the following principle to prune the candidate space:

> *The subset candidates of a candidate episode that have the same episode occurrences as the parent episode (the episode sets of the candidate episodes are equivalent) do not need to be generated as candidates.*

Because these subset patterns are shorter in length, but have the same frequency and periodicity as their parent, their compression ratio cannot be greater than their parent's value. Our pruning method generates patterns of shorter lengths from longer ones, which is essentially the opposite of an Apriori approach [1] where larger itemsets are generated from smaller ones.

An example of the algorithm incrementally generating candidates is shown in Table 4. For simplicity, in the example it is assumed that one maximal episode is identified each day, and the timestamps have been omitted. The creation of the first maximal episode generates one candidate, *{a, b, c, d}*. The second maximal episode results in the generation of two additional candidates: *{a, b, c, e}* from the maximal episode, and *{a, b, c}* from the intersection of the two maximal episode. The last maximal episode results in the generation of four additional candidates. By inspection, it can be seen that *{a, b}* should be identified as a significant episode because it occurs every day. In the example, we see that *{a, b}* is indeed generated as a candidate. Notice that it is not necessary to generate *{a}* and *{b}* as candidates, because these subsets of *{a, b}* only occur in episodes that contain both *a* and *b*. The pruning technique effectively eliminates these and several other unnecessary candidates from consideration. It is also important to note that in Steps 2 and 3, only those candidates that are common across multiple episodes are generated. Candidates that need to be created because a candidate is selected as interesting are generated in Step 5.

Table 4. Generating Candidates.

Day	Maximal Episodes	List of Generated Candidates
1	(a, b, c, d)	{{a, b, c, d}}
2	(a, b, c, e)	{{a, b, c, d}, {a, b, c}, {a, b, c, e}}
3	(a, b, d, e)	{{a, b, c, d}, {a, b, c}, {a, b, c, e}, {a, b, d}, {a, b, e}, {a, b}, {a, b, d, e}}

4.3 Step 4: Compute compression ratios

The list of maximal episodes is walked, so that the episode set of each C_n is updated with those maximal episodes containing the symbolic pattern represented by the candidate. Once the episode assignments have been completed, the algorithm uses auto-correlation techniques to determine the best repeating interval sequence. In Table 1, we presented an example in which the interval sequence *{48, 48, 72}* repeats. By calculating auto-correlation values, it can be discovered that a pattern repeats and is of length three. The auto-correlation analysis is performed for the fine-grained and course-grained interval sequence. Once the search for an interval sequence has been completed, the individual entries are evaluated to identify any mistakes that occur in the repeating pattern. Then, a fine-grained, course-grained, and frequency compression ratio is computed. The ratio with the largest value is selected as the compression for that candidate. The algorithm steps are shown in Figure 1.

```
foreach maximal episode ε_i
   foreach candidate C_n that contains
   ε_i
      add ε_i to the episode set of
      candidate C_n
      foreach candidate C_m
         compute compression ratios
         select largest compression
         ratio
```

Figure 1. Steps for Computing Compression Ratios.

4.4 Step 5 and 6: Select and output interesting episodes

The candidates are sorted, and the algorithm greedily identifies an interesting episode by selecting from the sorted list the candidate with the largest compression ratio. The events represented by the interesting episode are marked. To avoid selecting overlapping candidates, the second and subsequent candidates are rejected if any of the events represented by the occurrences of the candidate are already marked. Once a candidate is selected, additional candidates are generated by subtracting the selected pattern from all remaining candidates containing the pattern, and generating the difference as additional candidates. This was discussed in Steps 2 and 3. In Table 4, we present an example of candidates that would be generated because they were subsets of multiple episodes. In Table 5, we show the additional candidates that would be generated if *{a,b}* were selected as interesting. These steps are repeated until all candidates have been processed. Candidates that are selected as interesting are output.

Table 5. Candidates Generated Upon Selecting *{a,b}*.

Selected Episode	Candidates	Additional Candidates
{a, b}	{{a,b,c,d}, {a,b,c}, {a,b,c,e}, {a,b,d}, {a,b,e}, {a,b}, {a,b,d,e}}	{{c,d}, {c}, {c,e}, {d}, {e}, {d,e}}

5. EPISODE DISCOVERY FEATURES

It has been empirically demonstrated [5] that ED detects symbolic patterns that exhibit periodicity, improves the performance of predictors in an intelligent environment, and operates efficiently. We now discuss additional algorithm features and other information output by ED.

5.1 Ordering Knowledge

ED can output ordering information about the pattern. Because ED tracks the different permutations of the pattern, statistics can be presented on how many permutations there are and how many times each permutation occurs. If a pattern were totally ordered, then ED would only output a single permutation. If it were unordered, then ED would output multiple patterns, with no pattern occurring significantly more often than another.

5.2 Discovering Event Folding Window Parameters

The compression ratios can be used to search for the optimal window span and capacity parameters. Based on our encodings, a window size search should start from a small window size and continue to the largest. As the window size is increased, patterns of increasing length will be evaluated. Because the encodings favor longer patterns, the overall compression should also increase because patterns of longer length will be discovered. At some point, the compression will stabilize as the optimum window size is discovered. Using a synthetic dataset representing inhabitant activities in an intelligent environment [5], we show the compression ratios plotted for various values of t_w in Figure 2. For this dataset, the compression ratio peaked at a window span of 20 minutes. The dataset was constructed such that all interesting interactions occur within a fifteen-minute time frame, which corresponds to the window span indicated by the plot. This same approach can also be used with the window capacity parameter.

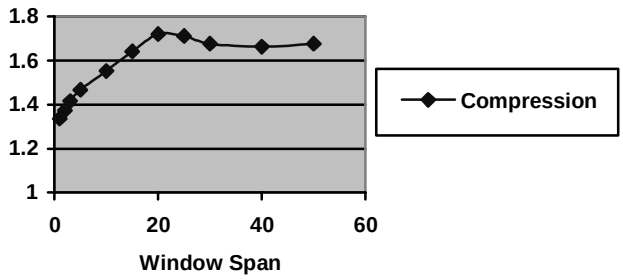


Figure 2. Window Span vs Compression Ratio.

5.3 Incremental Processing of Data and Knowledge Generation in Real-Time

Steps 1, 2, and 3 of the algorithm can be performed independently of steps 4, 5, and 6. Thus, the algorithm is able to incrementally process the events, and can be invoked when desired to produce the interesting episodes. This is important for a real-time system, because knowledge produce by the algorithm can be used at the same time the algorithm is collecting the necessary information to provide a knowledge update. In addition, it should be more efficient to generate knowledge based on an increment of data, rather than having to process the entire dataset.

To confirm our assumption, ED was run in an incremental manner by generating knowledge every month on a dataset consisting of intelligent environment events covering a nine-month period [5]. The following was observed:

- The exact same set of interesting episodes was generated once all of the input data was processed.
- It took thirty-four seconds to process the full nine months of activity data.
- It took eighteen seconds to incrementally process the ninth month. The processing time was reduced almost 50% by incrementally generating the knowledge.

5.4 Computing Interesting Episode Memberships

ED maintains the following statistical information on the interesting episodes:

1. For each interesting episode, ED maintains a list of the subsets that have been generated as candidates. Thus, we can quickly determine if a subset of a candidate has also been generated.
2. For each interesting episode, ED maintains a list of episodes that represent an occurrence of that symbolic pattern. Thus, we know how many occurrences there are of that symbolic pattern.
3. A collection of the episodes that have been generated by the event-folding window is maintained. Thus, we know the total number of episodes.

As events are processed, the contents of the current event-folding window maintained by ED can be retrieved. Using Bayes' rule and these statistics, it is possible to determine for each interesting episode the probability that the current event-folding window will contain this interesting episode. For example, if the window contains *{a,b,d,e}*, ED could report a probability of 82% that the window will eventually contain interesting episode *{a,b,c}*. Because ED incrementally processes the events, membership computation can be performed on-line as every event is processed.

6. INTEGRATING ED WITH A DECISION MAKER

We have implemented an intelligent environment architecture called MavHome [2]. The MavHome decision-making component uses the periodic episode and ordering knowledge output of ED to create a hierarchical hidden Markov model (HHMM) [3][9]. The system controls the environment by automating the most likely event based upon the current observation. This model is built from multiple passes through the dataset, which creates an increasing hierarchy of abstraction. Statistical information collected and derived through knowledge discovery is used to define the horizontal and vertical transitions of the model.

Experiments using simulated environment data have proven the ability of this approach to automatically create HHMMs from inhabitant interaction data. We have created datasets [5] containing months of inhabitant data based on stochastic simulation, organizing the data into human-perceived episodes (e.g., watching TV, entering room, exiting room, reading), and then using a simulator to distribute activities over specified periods of time. For example, we establish activity patterns that show someone entering the living room, watching TV, going to the kitchen, leaving, entering, and so forth all distributed over specified times of occurrence over a specified period. A test dataset was generated that contains twenty-three embedded behaviors. ED processed the data and found thirteen periodic episodes that correspond to those various environmental activities. A HHMM was automatically constructed, and it was manually verified that it correctly encoded the ED data. We are continuing our efforts in this area, and will eventually perform a comparison of our approach with other techniques.

7. CONCLUSIONS

In this paper, we have shown that the ED algorithm automatically detects regularity intervals, provides statistics on pattern ordering, and computes interesting episode membership values. We have also demonstrated that ED supports discovery of the parameters for the event-folding window, can be operated in an incremental manner to support real-time environments, and can be used to bootstrap the states of a decision maker.

In our future work, we intend to evaluate adding ordering as a characteristic of the encodings. We also are investigating using the compression ratios to understand drift and shift. In addition, we will investigate temporal aspects of the membership calculation in order to provide even more information on the likelihood of an episode occurrence. Finally, we will continue the integration of ED with a decision maker, and ultimately incorporate the technique into a setting with live inhabitants.

8. REFERENCES

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 11th International Conference Data Engineering (ICDE 1995)*, pp. 3-14, Taipei, Taiwan, March 1995.

[2] S. Das, D. Cook, A. Bhattacharaya, E. Heierman, and T. Lin. The Role of Prediction Algorithms in the MavHome Smart Home Architecture. *IEEE Wireless Communications*, vol. 9, no. 6, pp. 77-84, December 2002.

[3] Shai Fine, Yoram Singer, and Naftali Tishby. The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning*, 32(1):41– 62, 1998.

[4] J. Han and M. Kamber. *Data Mining*. Morgan Kaufman Publishers, 2001.

[5] E. Heierman and D. Cook. Improving home automation by discovering regularly occurring device usage patterns. In *Proc. 3rd International Conference on Data Mining (ICDM'03)*, pp. 537-540, Melbourne, FL, November 2003.

[6] H. Mannila, H. Toivonen, and A. Verkamo. Discovering frequent episodes in sequences. In *Proc. 1st International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pp. 210-215, Montreal, Canada, August 1995.

[7] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific Publishing Company, 1989.

[8] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the Fifth International Conference on Extending Database Technology* (EDBT-96), pp. 3-17, Avignon, France, 1996.

[9] G. Theocharous, K. Rohanimanesh, and S. Mahadevan. Learning Hierarchical Partially Observable Markov Decision Processes for Robot Navigation, 2001. IEEE Conference on Robotics and Automation.

An Efficient Method For Finding Emerging Large Itemsets

Susan P. Imberman
College of Staten Island
The Graduate Center
City University of New York
2800 Victory Blvd.
Staten Island, NY 10314
Imberman@postbox.csi.cuny.edu

Abdullah Uz Tansel
Baruch College
The Graduate Center
151 East 25th Street
New York, NY 10010
tansel@baruch.cuny.edu

Eric Pacuit
The Graduate Center
City University of New York
365 Fifth Avenue
New York, NY 10016
e_pacuit@hotmail.com

ABSTRACT
The incremental mining of association rules has been shown to be more efficient than rerunning standard association rule algorithms such as Apriori. As each increment is processed, we see the emergence of some itemsets. An itemset that has emerged is one that was small and is large in the current increment. An emergent large itemset is a small itemset that has the potential to become large, and will do so with high probability. In this paper we modify an existing incremental algorithm, UWEP, so that it can identify emergent large itemsets. We show that, on average, 65% of the emergent large itemsets identified by the algorithm actually do emerge.

General Terms
Algorithms, Experimentation

Keywords
Incremental Algorithms, Association Rules, Negative Border, Temporal Association Rules, Emergent Large Itemsets.

1. INTRODUCTION
The use of association rules to find the co-occurrence between variables in large datasets is a well researched technique for data mining. Association rules have been used in many domains, most notably market basket data. Here, mined associations between store products have been used for product shelf placement and customer marketing. In the banking industry association rules are used to find the characteristics of high risk credit customers.

The problem of finding association rules can be decomposed into two steps. The first step involves finding all large itemsets. This step is the most difficult. Algorithms that accomplish this, such as Apriori [1,2], are exponential in the worst case. Once the large itemsets are found, generating association rules is straightforward. and can be accomplished in linear time.

TDM '04, August 22, 2004, Seattle, Washington.

When a database is expanded with an incremental increase in the number of transactions, we can either rerun an algorithm such as Apriori, or use the knowledge obtained from the "original" database to reduce the processing time for finding small itemsets that have become large or any large itemsets that have become small. Several incremental algorithms have been proposed that can efficiently find large itemsets. [4,5,6,7,9,11]. UWEP (Update With Early Pruning) has been shown to be an efficient algorithm that addresses the incremental association rule problem. UWEP gains its efficiency by pruning from consideration, those large itemsets in *DB* (the original database), that are not present in *db* (the increment) and become small due to the change in the overall number of transactions in the combined "new" database, *DB+db*. UWEP scans DB at most once and db exactly once. It has been shown to generate and count the minimum number of candidates in order to determine the new set of association rules. [4,5]

The negative border has been defined as the set of minimal small itemsets, closed with respect to set inclusion, and whose subsets are all large. [12] The negative border is a subset closed collection of itemsets. Thomas, Bodagala, Alsabti, and Ranka [11] used the negative border set to see which itemsets have changed status from large to small and vice versa. Dong and Li [8] use interval closed borders to find emergent patterns. They define an emergent pattern as an itemset whose support increases significantly between databases.

If we look at emergent patterns with respect to the negative border we can see whether a small itemset is becoming large. In this paper we use this property of the negative border, in a modified version of UWEP called NUWEP (Negative border Update With Early Pruning), to find these emergent large itemsets. We view emergent large itemsets as a special case of the emergent patterns described by Dong and Li. An emergent large itemset (ELI) can be thought of as an itemset that is currently small in the combined database but has increased its support such that it will eventually become large. ELI are particularly interesting emergent patterns. For example, in the market basket domain, if we define an interval as the time between wholesale purchases, recognizing that a set of items will emerge or become large in the next time period, can allow the storekeeper to order these items much earlier than usual. Thus the storekeeper can avoid being short these items and losing the income their sales could have generated. Finding emergent large itemsets does not add a lot of increased overhead to the standard incremental algorithm. Our contribution in this paper is to show how an incremental approach allows us to find ELIs efficiently. We can identify ELIs that have the potential to

emerge with results that are better than random. Moreover, our incremental approach allows us to identify the time an itemset becomes large before waiting for the next batch cycle.

The organization of this paper is as follows: Section 2 gives the definition of the emergent large itemset problem. Section 3 outlines our approach for solving the emergent large itemset problem. In section 4 we show experimental results derived from synthetic data. Section 5 is our discussion and section 6 our conclusions.

2. STATEMENT OF PROBLEM

Define the support of an itemset *I,* as the number of transactions containing that itemset, divided by the number of transactions in the dataset. Itemsets with support above a user defined threshold called *minsup* are called large itemsets [1,2,3]. Given a set of items *I*, such as products in a store, an association rule is an implication $X \Rightarrow Y$ where *X* and *Y* are subsets of *I* and $X \cap Y = \phi$. Interesting association rules are defined by two metrics, support and confidence. Support is a measure of a rule's significance with respect to the database, and confidence is a measure or the rule's strength.

More formally, let *N* be the number of transactions in a database *DB*. Given an association rule $X \Rightarrow Y$, let S_1 be the number of transactions in *DB* that satisfy *X*. *X* satisfies a transaction *t* in *DB* if the items contained in *X* are also in *t*. Let S_2 be the number of observations in *T* that satisfy the vector $X \cup Y$. The support *s* of $X \Rightarrow Y$ is S_2 / N and the confidence *c* of $X \Rightarrow Y$ is S_2 / S_1 . An interesting association rule has support above or equal to a user defined parameter called *minsup*, and confidence equal to or above a user defined parameter called *minconf*. Large itemsets can generate interesting association rules. Finding large itemsets is difficult and costly, most times exponential. Finding rules from a large itemset is linear. Therefore most methods for finding association rules concentrate on efficiently finding large itemsets.

Many association rule algorithms have been formulated to address this problem. [1,2,3]. The problem increases in cost when one wishes to find large itemsets in a database that is increasing in size. Rerunning the association rule algorithm is costly, especially since much of the cost in processing large datasets lies in parsing the data. Incremental algorithms take advantage of the knowledge gained from the previous increment, to reduce iterations over the entire database. UWEP (Update With Early Pruning) is an efficient incremental algorithm, that counts the original database at most once, and the increment exactly once. In addition the number of candidates generated and counted is minimum. [4,5].

As time progresses, we see many interesting patterns with regard to the change in status of individual itemsets. An itemset that was small can become large, large itemsets can become small, or an itemsets may remain large or small. We define small itemsets that are moving toward large as emerging. Conversely, large itemsets moving toward small are submerging. A small (large) itemset that becomes large, i.e. support is above (below) *minsup*, is said to have emerged (submerged). The problem we address in this paper is, can we identify itemsets that are currently emerging (submerging). Next, which of these itemsets have the potential to emerge (submerge) within the next increment. Or, more generally, can this happen within *n* intervals. For this paper we talk about emergence, but the solution for the submergence problem is analogous.

For the remainder of this paper we use the following notation:

DB is the set of old transactions (where transactions denote the records in the original database)
db is the set of new coming transactions (the increment)
DB+db is the set of old and new coming transactions
$support_{DB}$ (X) is the support of itemset X in *DB*
$support_{db}$ (X) is the support of X in *db*,
$support_{DB+db}$ (X) is the support of X in *DB+ db*
tidlist *DB* (X) is the transaction list of X in *DB*
tidlist *db* (X) is the transaction list of X in *db*
tidlist *DB+db* (X) is the transaction list of X in *DB + db*
C_{db}^{k} is the set of candidate k-itemsets in *db,* where *k* is the number of items in that itemset.
L_{db}^{k} is the set of large k-itemsets in *db*
L_{DB}^{k} is the set of large k-itemsets in *Db*
L_{DB+db} is the set of large itemsets in *DB + db*.

2.1 Emergent Large Itemsets

In *Figure1* we have modified the formalizations given by Dong and Li in [8] for the special case of ELI. *Figure1* partitions the space of itemsets. It can also be visualized as all the possible transitions for an itemset X from *DB* to *DB + db*.

Definition 2.1.1: The support count (*SC*) of an itemset is the number of transactions that an itemset satisfies.

Assume that *db* is of constant size and is smaller than *DB*. Even though we place constraints on the problem, the following easily generalizes to the case where *db* varies. *Figure1* plots the support count in *DB* (denoted as SC_{DB}) against the support count in *db* (denoted as SC_{db}). Each point in the graph depicts an ordered pair (SC_{db}, SC_{DB}) where the sum of SC_{db} and SC_{DB} is an itemset's support count in *DB+db* at some increment interval.

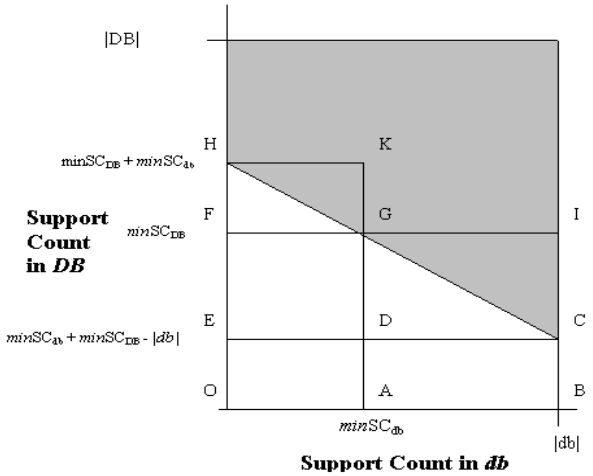


Figure 1 Emergent Itemsets

Definition 2.1.2 $minSC_{Db+db}$ is the minimum number of transactions needed to be large in *DB+db*.

Definition 2.1.3 $minSC_{DB}$ is the minimum number of transactions needed to have to have been large in *DB*.

Definition 2.1.4 $minSC_{db}$ is the minimum number of transactions needs to have been large in *db*.

If the increment adds no transactions to an itemset's support count, then in order to achieve $minSC_{Db+db}$ it's support count in *DB* has to be equal to $minSC_{DB} + minSC_{db}$. This is point H in *Figure1*. Alternately, if an itemset's *SC* is equal to $|db|$ in *db,* then to be large it's support in *DB* has to be some $SC = n$, where $n > 0$, and $n = minSC_{DB+db} - |db|$. This is point C in *Figure 1*.

Lemma 2.1.1: All points G = (SC_{db}, SC_{DB}), where $SC_{db} + SC_{DB} = minSC_{Db+db}$ lies on line HC.

Proof: : Proof directly follows from the definition of the line HGC. At point H, SC_{DB} of an itemset X has sufficient support at *DB* to become large at *DB+db* without any contribution from *db*, i.e., $SC_{DB} = minSC_{Db+db}$. From point H to point G, SC_{DB} declines, however the decline is compensated by SC_{db} and their sum ($SC_{DB} + SC_{db}$) equals to $minSC_{Db+db}$. At point G, an itemset X has the exact support count at both *DB* and *db* and hence G is on the line HC. For any point on GC, the increase in SC_{db} compensates the decrease in SC_{DB} and their sum still equals, $SC_{DB} + SC_{db} = minSC_{Db+db}$. □

Lemma 2.1.2 For all points G described in Lemma 2.1.1on the line HC, (SC_{db}, SC_{DB}) where $SC_{db} + SC_{DB} = minSC_{Db+db}$

Proof: Proof directly follows from the Lemma 2.1.1. (SC_{db}, SC_{DB}) = $minSC_{Db+db}$ is true by the definition of line HC and $minSC_{DB} + minSC_{db} = minSC_{Db+db}$ holds by the definition of minsup over *DB + db*. □

Line HC partitions the space of all itemsets in *DB+db* into large and small. The shaded area in *Figure1* represents all the large itemsets and it includes Line HC. In fact, using defined values for $minSC_{Db+db}$, $minSC_{DB}$, ,and $minSC_{db}$ we can further partition the itemset space as shown in *Figure1*. Each partition exhibits some interesting properties with respect to itemsets in *DB*, *db,* and *DB+db*. Specific partitions under HC contain itemsets that are emerging in the current increment. For example, the area defined by ΔHFG represents those itemsets that were large itemsets in *DB*, small itemsets in *db*, and now are small in *DB+db.* These itemsets have therefore submerged. ΔGIC represents itemsets that were small in *DB* and large in *db*. These itemsets have emerged.

Lemma 2.1.3: An itemset that is small over *DB* with support *n,* and large in *db* with support >= *minsup,* has support in *DB+db* >*n*.

Proof: : Let n = SC_{DB} / |DB|. SC_{db} / $|db|$ >= *minsup* is given. Since the itemset is small in *DB*, $n < minsup$. We can break SC_{db} into j + k such that j / SC_{db} = n. It is shown that (SC_{DB} + j) / ($|DB|$ + $|db|$) = *n* [11]. Clearly adding a positive value to the numerator increases the value of the fraction, i.e., (SC_{DB} + j + k) / ($|DB|$ + $|db|$) > *n*. □

Lemma 2.1.4: An itemset that is small *DB* with support *n,* and small in *db* with support < *n* < *minsup,* has support in *DB+db* < *n*.

Proof: Proof is analogous to the proof of Lemma 2.1.3. □

Therefore, according to Lemma 2.1.3 and 2.1.4, all itemsets in area ABCG are emerging in the current interval and all itemsets in area OAGH are submerging. To find all ELI's in the current interval, we need to identify all itemsets in ABCG.

2.2 Interesting Emerging Itemsets

Finding interesting ELI's can be both a subjective or objective problem. For example, if we look at an itemset, that contains gardening tools such as hoe, rake, and spade, an ELI containing these items wouldn't be interesting if it were emerging during the spring/summer seasons. It would be interesting if it were emerging in the winter season. This would be a subjectively interesting ELI. Subjectively interesting ELI are domain and user dependent.

Definition 2.2.1 The growth rate of an ELI is the average gain in support over *n* increments.

By definition, ELI's that continue to remain ELI's will eventually become large. If the ELI is growing at a very small rate, then it will become large at some time far into the future. ELI's with faster growth rates will emerge sooner. Therefore, one type of objectively interesting ELI is one that will emerge within the next increment, or within the next *N* increments.

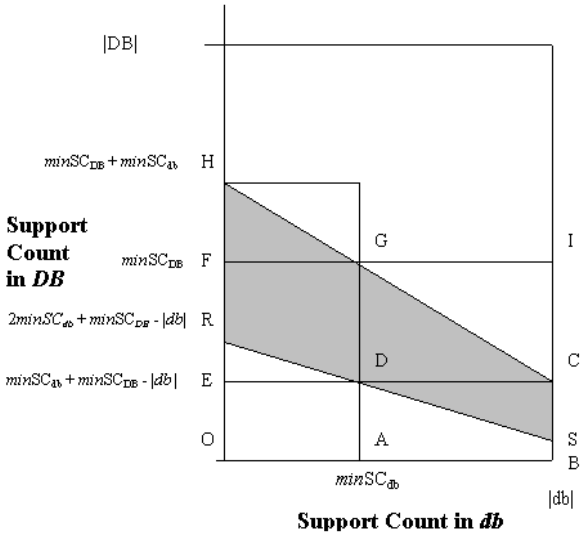


Figure 2 Potentially Emergent Large Itemsets

Maintaining our assumptions about a constant increment size and |db| < |DB|, we can establish a lower bound on the support count needed by an itemset in the current increment interval to achieve emergence in the next increment. The largest possible gain in support count in the next increment is equal to $|db|$. This occurs when an itemset is contained in every transaction in the increment. To have the *potential* to emerge in the next increment, the support count of the itemset in *DB+db* needs to be greater than or equal to $2minSC_{db} + minSC_{DB} - |db|$ in the current increment. All points with this value are represented by line RS in *Figure 2*. For example, if we have a $|DB| = 10{,}000$, $|db| = 1{,}000$ and $minsup = .20$, then the minimum support count for the current

increment is 2,200 (2,000 from *DB* + 200 from *db)*. If an itemset can add the maximum support incremental support count, a total of 1,000 from *db*, in the next increment, it would need a support count of at least 1,400 in the current increment to be able to attain the minimum support count of 2,400 needed to become large. The band of itemsets between line RS and line HC are all itemsets that have the potential to become large in the next increment, by this formula. If we use only information from the current increment to determine which ELI's have the potential to become large in the next increment, then all itemsets that are in this band and are emerging have the highest probability of emerging in the next increment. Therefore, itemsets in GDSC are most likely to emerge in the next increment.

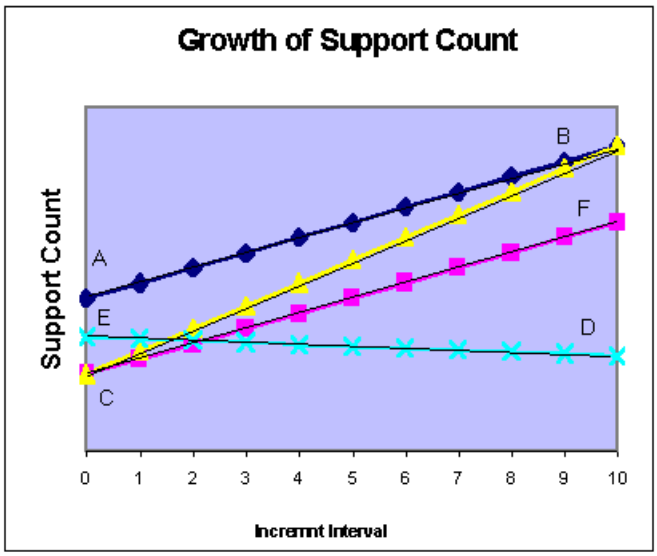


Figure 3 Growth in Support of Itemsets

In large datasets this can be a large group. We can bound this set even further by looking at an itemset's growth rate. If we were to plot the support count of an itemset over time, we see that in order to maintain minimum support, an itemset has to have a constant growth rate. Line AB in *Figure 3* represents the minimum support count, in the cumulative database (*DB*+*db*), in order for that itemset to remain at exactly *minsup*. If a small itemset's support count increases at the same rate as AB, i.e. its support in *db* is *minsup* for each interval, then that itemset will never become large. This is shown by line CF. Note the slope of CF in the graph is the same as AB. Those itemsets whose growth decreases with time, as shown by ED have slope less that of line AB. Emergent large itemsets, such as CB have slope greater than AB. Thus, even though an itemset is emerging, it may not emerge in the next increment or within the next *n* increments. Those itemsets whose growth rates allow them to become large in the next interval are potentially emergent large itemsets. Therefore these itemsets are those itemsets whose rate of growth is larger than that of an itemset maintaining *minsup*.

3. NUWEP, AN ALGORITHM FOR FINDING INTERESTING EMERGING LARGE ITEMSETS

Keeping track of all small itemsets' growth rates can be exponentially large in both time cost and memory cost. The negative border is a concise way to represent larger sets of small itemsets. The negative border is the set of small itemsets whose supersets are large [12]. Thomas, Bodagala, Alsabti, and Ranka [11] proved that an itemset *X* that was previously small in the original dataset and large in the *"new"* or original plus incremental dataset, has either moved from the negative border set to the set of large itemsets in the new dataset, or some subset of *X* has moved from the negative border set to the set of large itemsets in new dataset.

Lemma 3.1: An itemset in the negative border will emerge at the same time or before any of its supersets.

Proof: Let *X*, *Y* be itemsets, *X* is in the negative border and *Y* be an immediate superset of *X*. Lets assume that *Y* becomes large and *X* is not large. Apparently, $support_{DB}(X) < minsup$ and $support_{DB+db}(X)$ but $support_{DB}(Y) > minsup$. This is a contradiction since all the subsets (i.e., *X*) of a large itemset (*Y*) are also large [8]. □

Lemma 3.2 The growth rate of an itemset in the negative border will be equal to or greater than any of its supersets.

Proof: Let *X*, *Y* be itemsets, X is in the negative border and *Y* be an immediate super set of *X*. Lets assume that growth rate of *Y* is larger that the growth rate of *X*, i.e. $support_{DB+db}(Y) > support_{DB+db}(X)$. From the definition of support, this implies $SC_{Db+db}(Y) > SC_{Db+db}(X)$ and it is clearly a contradiction. □

If we keep track of the growth rate of an itemset in the negative border, and that itemset has the potential to emerge in the next increment, then we can easily find the supersets that will also emerge.

3.1 UWEP to NUWEP

UWEP is an incremental association rule algorithm that has been shown to efficiently solve the incremental association rule problem. It has been shown to generate and count the minimum number of candidates in order to determine the new set of association rules. [4,5]. To find ELI, we can modify UWEP so that in addition to counting and keeping track of large itemsets, it keeps track of the negative border.

Appendix 1 contains UWEP with the additional steps necessary to maintain the negative border. We call the modified algorithm, NUWEP (Negative border Update With Early Pruning).

Lemma 3.1.1**:** The number of candidates generated and counted by NUWEP algorithm is minimum.

Proof**:** Candidate generation is driven by the increment database, *db*. Therefore it would suffice to show that we generate and count minimum number of candidates in *db* at each level since NUWEP is a level wise algorithm. C^1_{db} contains only the itemsets whose support is greater than zero and it is a minimum bound. At level k only the itemsets that are large in db and DB + db are placed into L^k_{db}. Itemsets in L^k_{db} are used in generating C^{k+1}_{db}. Clearly this is the minimum possible number of candidate itemsets. The

candidates counted in *db* are minimum, since itemsets that are large in *DB* or *db* are only considered.

The number of itemsets counted over *DB* is also a minimum since only itemsets that are small in *DB* and large in *db* need to be counted over *DB*. If the itemset is in NB_{db}^{k} we do not need to count it. However, if NB_{DB}^{k} expands, itemsets need to be counted. There are two possibilities:

1) Any small itemset whose subsets are large over *DB*+ *db* is in NB_{DB+db}^{k+1}, and it needs to be counted if it is small in DB.
2) 2) Consider an itemset X that is large in *DB*+*db* and large in *db,* but small in *DB*. Consider an itemset Y that is small over *db* but large over *DB*+*db* and *DB*. An immediate superset of *X* made up of the items in *XY* is in the NB_{DB+db}^{k+1}. An algorithm that considers only counting these itemsets when generating the negative border is minimal. Since NUWEP generates the negative border and counts only itemsets that are of the two possibilities mentioned above, NUWEP is minimal. □

Of course maintaining the negative border adds cost with respect to time and memory. The memory cost of the negative border is much smaller than maintaining the same information on all small itemsets since the memory required to maintain both large and small itemsets is exponential with respect to the dimensionality of the dataset. This is not tractable for most large datasets.

As for time costs, we ran several experiments comparing execution time of NUWEP to UWEP with synthetic data generated as per [3]. We generated a 10,000 transaction dataset where the average transaction was 5 items, the average large itemset was 4 items, and the total number of items was 10 (T5.I4.D10). The results of these experiments on this dataset are in *Figure 4*. We see that that as we increase the size of *db* the difference between UWEP and NUWEP's execution time is constant. In fact for low levels of support the difference is negligible. In addition overall execution time increases at a logarithmic rate. As we increase the size of *DB*, holding *db* constant, we see that the difference between NUWEP and UWEP is also constant. Hence the cost in maintaining the negative border is related to increases in size of *db* rather than *DB*. We repeated these experiments with a dataset T5.I2.D10 with similar results. The largest negative border equals C(n, n/2), which is the combination of n items taken n/2 at a time, and n is equal to the total number of items. Therefore there is a bound on how large the constant difference between the two algorithms can be.

3.2 Finding potentially Emergent Large Itemsets.

Modifications were made to NUWEP that allow the calculation of growth rate for itemsets in the negative border. These modifications were minor. Itemsets were always stored in data structures that include their support count. Therefore, because of this, we know the support count of the negative border itemset in DB for the current interval. The algorithm update finds the updated value of the support count in *DB+db.* From these two values we calculate the growth rate, which is the slope of the line that connects these two points. Note that the additional memory required for this approach is a numeric variable to record the growth rate. Additional memory is needed only for those itemsets in the negative border.

More formally, given an itemset whose support count in *DB* is SC_{DB} and its support count in *DB+db* is SC_{DB+db}, then the growth rate of that itemset is $SC_{DB+db} - SC_{DB}$. The growth rate of an itemset that maintains minimal support is, $minSC_{Db+db} - minSC_{DB}$. An itemset where $\frac{SC_{DB+db} - SC_{DB}}{\min SC_{DB+db} - \min SC_{DB}} > 1$ is an emerging itemset. An itemset needs a support count of at least $minSC_{Db+2db}$ to emerge in the next increment. A potentially emerging large itemset is one that is emerging and $SC_{DB+db} + (SC_{DB+db} - SC_{DB}) > minSC_{Db+2db}$. Once an itemset in the negative border is identified as a potential ELI, then we can easily find all supersets that are also potentially emerging large itemsets.

4. EXPERIMENTAL RESULTS

We used the two synthetic market basket datasets that was created in for the UWEP/NUWEP comparison in section 3. NUWEP was run in turn on each of the datasets. In addition to the negative border, we kept track of the set of potential ELI, and those itemsets that actually emerged in the next increment. *Table 1* shows the results of these experiments.

Table 1 Potentially Emergent Large Itemsets

INCREMENT	#Potentially Emergent Large Itemsets	# That Emerged In Next Increment	Percent Of Actual Large Itemsets That Were Potentia
1	62	51	82
2	44	33	75
3	56	49	87
4	48	37	77
5	76	70	92
6	43	39	90
7	51	48	94
8	38	37	99

As can be seen from *Table 1,* for each increment, a large percentage of the itemsets that did emerge were correctly identified. The significance is that these itemsets were identified in the period prior to the one in which they actually did become large.

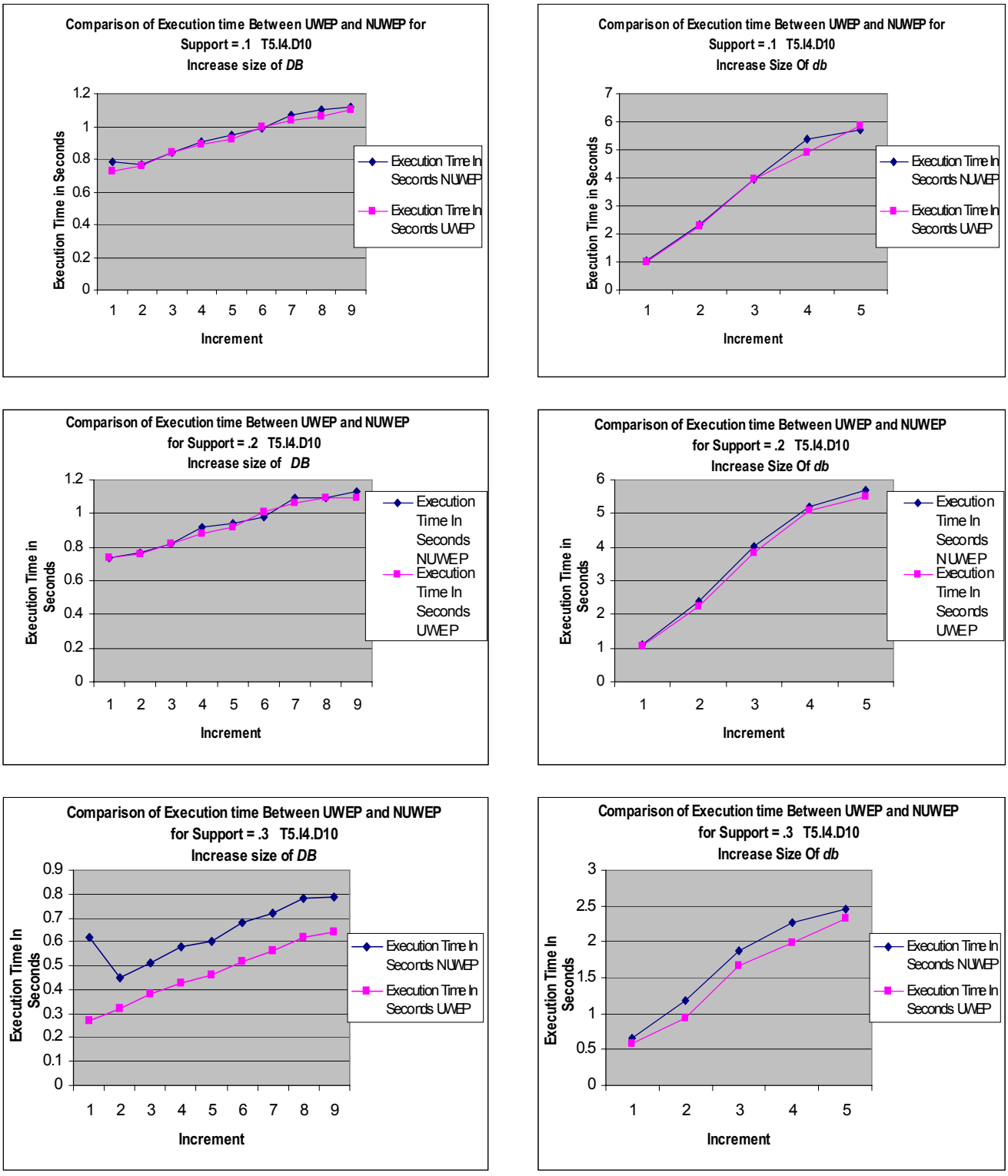


Figure 4 Comparison of UWEP to NUWEP

5. DISCUSSION

In the past, the incremental association rule algorithms were created to gain efficiency over standard association rule algorithms when finding large itemsets. Our approach shows that at the risk of losing some efficiency, we can gain more information about itemset behavior. By watching an itemset and its supersets move from the negative border to large, and possibly back indicate that there are patterns that can yield interesting information. For example, it is quite possible that a group of itemsets will oscillate between being large and ΔHFG in *Figure 2*. If this happens often, to a significant number of itemsets, one could conclude that support may be set either too high or too low. Since support is a user defined parameter, this oscillation behavior may give a heuristic for fine tuning this parameter.

Although we have shown how we can predict, with reasonable accuracy, which itemsets will emerge in the next increment, it is only natural to ask how one might predict if an itemset will emerge within the next n increments. If we use only the information available to us in the current increment, then this can be easily determined by adjusting the calculation of an emerging itemset. An itemset that will potentially emerge within n increments, is an itemset that is currently emerging and $SC_{DB+db} + n(SC_{DB+db} - SC_{DB}) > minSC_{Db+ndb}$. Of course, the larger n is, the less accurate our predictions.

Besides extending our view from one increment to n increments, we can also utilize the knowledge obtained about itemset behavior from past increments, including the current increment, to predict emergence. The problem here is that processing large databases requires substantial amounts of memory to store just the information on large itemsets and the small itemsets in the negative border. Maintaining too much information from past increments can be very memory intensive. If memory is not an issue, then we can remember the support counts of each itemset in the negative border, for each increment. These points may contain a rich set of interesting patterns. This allows us to apply standard statistical prediction techniques to these points to find the growth rate of an itemset. By doing this, we can avoid labeling itemsets that have spikes in support count as being potentially emergent. Thus, we may see an increase in the accuracy of prediction. Since memory is usually an issue, we can get a "dirtier" picture of an itemset's potential by time stamping the itemset with the increment number within which it enters the negative border, remembering the support count of the itemset when this happens, and use this information with the information in the current interval to determine the growth rate. The experimental results showed that we could predict, within the current period, a significant number of itemsets that would become large in the next period. The number of itemsets identified as being potential, and did emerge was less significant. We feel that if we were to use more past history, as outlined above, we could reduce the number of potentially large itemsets while still predicting well for the next increment.

6. CONCLUSIONS

In this paper we discussed a method that identifies small itemsets that have the potential to become large in the next increment. Our method makes use of an incremental approach, and by doing this decreases the time needed for processing if standard association rule algorithms did the same. Our results indicate that a good percentage of the itemsets that we predict to emerge actually do emerge. Our methodology easily extends to predicting emergence within a set number of increments. For the future, we would like to investigate using more history to identify potentially emergent large itemsets and compare them to the approach used in this paper.

7. REFERENCES

[1] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In Proceedings of 1993 ACM SIGMOD Intl. Conf. on Management of Data, pages 207--216, Washington, D. C., May 1993.

[2] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In Usama Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramaswamy Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, pages 307--328. AAAI/MIT Press, 1996.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Proceedings of 20 th Intl Conf. on Very Large Databases (VLDB'94), pages 487--499, Santiago, Chile, 1994.

[4] N.F. Ayn, A.U. Tansel, and E. Arun. An efficient algorithm to update large itemsets with early pruning. Technical Report BU-CEIS-9908 Dept of CEIS Bilkent Uniiversity , June 1999.

[5] N.F. Ayn, A.U. Tansel, and E. Arun. An efficient algorithm to update large itemsets with early pruning. Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, August 1999.

[6] David Wai-Lok Cheung, Jiawei Han, Vincent T. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental update technique. In Proceedings of Intl. Conf. on Data Engineering (ICDE'96), New Orleans, Louisiana, February 1996.

[7] David Wai-Lok Cheung, Sau Dan Lee, and Benjamin Kao. A general incremental technique for maintaining discovered association rules. In Proceedings of the 5 th Intl. Conf. on Database Systems for Advanced Applications (DASFAA'97), Melbourne, Australia, April 1997.

[8] Guzhu Dong, Jinyan Li. Efficient Mining of Emerging Patterns: Discovering Trends and Differences. Proceedings of the Fifth ACM SIGKDD International Conference on

Knowledge Discovery and Data Mining, San Diego, August 1999.

[9] N. L. Sarda and N. V. Srinivas. An adaptive algorithm for incremental mining of association rules. In Proceedings of DEXA Workshop'98, pages 240--245, 1998.

[10] A. Savasere, Edward Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In Proceedings of 21 st Intl. Conf. on Very Large Databases (VLDB'95), Zurich, Switzerland, September 1995.

[11] Shiby Thomas, Sreenath Bodagala, Khaled Alsabti, and Sanjay Ranka. An efficient algorithm for the incremental updation of association rules in large databases. In Proceedings of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining (KDD'97), New Port Beach, California, 1997.

[12] Hannu Toivonen. Sampling large databases for association rules. In Proceedings of 22nd Intl. Conf. on Very Large Databases (VLDB'96), Mumbay, India, September 1996.

Appendix I - NUWEP algorithm

DB is the set of old transactions (where transactions denote the records in the original database)
db is the set of new coming transactions (the increment)
DB+*db* is the set of old and new coming transactions
$support_{DB}$ (X) is the support of itemset X in *DB*
$support_{db}$ (X) is the support of X in *db*,
$support_{DB+db}$ (X) is the support of X in *DB*+ *db*
tidlist *DB* (X) is the transaction list of X in *DB*
tidlist *db* (X) is the transaction list of X in *db*
tidlist *DB*+*db* (X) is the transaction list of X in *DB* + *db*
C^k_{db} is the set of candidate k-itemsets in *db,* where *k* is the number of items in that itemset.
L^k_{db} is the set of large k-itemsets in *db*
L^k_{DB} is the set of large k-itemsets in *Db*
L_{DB+db} is the set of large itemsets in *DB* + *db*.
N_{DB}, is the negative border of *DB*
NB_{DB+db} is the negative border of *DB*+*db*
NC^k_{DB+db} are the candidate *k- itemsets* for the negative border

NUWEP(*DB*, *db, N_{DB}, L_{DB},* |*DB*|, |*db*|, *minsup*);
1 C^1_{db} = all 1-itemsets in *db* whose support is greater than 0
1a N_{DB+db} = Ø; *BorderSet* = Ø // Initialize N_{DB+db}
2 *PruneSet* = L^1_{DB} - C^1_{db}
3 **while** *PruneSet* ≠ Ø **do begin**
4 X = first element of *PruneSet*
5 **if** $support_{DB}$ (X) < *minsup* * | *DB* + *db* | **then begin**
6 remove X and all supersets from L_{DB} *and PruneSet*
6a remove all of X's supersets from N_{DB}
6b add X to N_{DB+db}
6c **end**
7 **else**
8 **begin**
9 add the supersets of X in L_{DB} to the *PruneSet*
10 add X to L_{DB+db}
11 remove X from L_{DB}
12 **end**
13 remove X from *PruneSet*
14 **end**
15 k = 1

```
16    while C^k_db ≠ Ø or L^k_DB ≠ Ø or BorderSet ≠ Ø do begin
17        Unchecked = L^k_DB
18        for all X ∈ C^k_db do
19            if support_db (X) < minsup * | db|  then        // X is small in db
20                if X ∈ L^k_DB then begin                    // X is large in DB
21                    remove X from Unchecked
22                    if support_DB + db (X) < minsup * |DB + db| then begin
                                                              // X is small in DB + db
23                        remove all supersets of X from L_DB
23a                       remove all supersets of X from NB_DB
23b                       add X to N_DB+db
23c                   end
24                    else                                    // X is large in DB + db
25                        add X to L_DB+db
26                end
26b               else add X to NB_DB+db
27            else                                            // X is large in db
28                if X ∈ L^k_DB then begin                    // X is large in DB
29                    remove X from Unchecked
30                    add X to L_DB+db
31                    add X to L^k_db
32                end
33                else begin                                  // X is small in DB
34                    find support_DB (X) using tidlists if not in N_DB
35                                                          // X is large in DB + db
36                    if support_DB + db (X) >= minsup * | DB + db| then begin
37                        add X to L_DB+db
38a                       add X to BorderSet  // X became large and the negative
                                              // border expands
38c                       remove X from N_DB
38d                       end
38e                   else add X to NB_DB+db
40                end


41        for all X ∈Unchecked do begin      // X is large in DB but not counted in db
42            find support_db (X) using tidlists
43            if support_DB + db (X) < minsup * |DB + db| then begin  // X is small in DB + db
44                remove all supersets of X from L_DB
44a               add X to N_DB+db
44b               remove all ~~subsets~~ supersets of X from N_DB
44c           end
45            else                                            // X is large in DB + db
46                add X to L_DB+db
47        end
47a       for all X ∈ BorderSet do begin     // X is large in db but not combined with small itemsets in
                                             DB that are large in DB+db
47b           NC^k_DB+db = Generate all k + 1 supersets of X  //Candidate itemsets for Negative
                                                              //Border.
47b2          remove X from BorderSet
```

47b3 **end**

47c **for** all X ∈ NC^{k}_{DB+db}

47d **if** X has subsets in N_{DB+db} remove X from NC^{k}_{DB+db}

47e **else if** $support_{DB+db}$ (X) < *minsup* * |DB + db| do begin

47f add X to N_{DB+db}

47g remove X from NC^{k}_{DB+db}

47h **end**

47i **else do begin**

47j add X to L_{DB+db}

47k add X to *BorderSet*

47k2 remove X from NC^{k}_{DB+db}

47l **end**

47m **end**

48 k = k + 1

49 $C^{k}_{db} = generate\_candidate\ (L^{k-1}_{db})$ //generate candidate k - itemsets

50 **end**

50a **for** all X ∈ N_{DB} **do begin**

50b // X is small in DB and db since large itemsets of db are already considered

50c add X to N_{DB+db}

50d **end**

Order and Mess in Text Categorization: Why Using Sequential Patterns to Classify

S. Jaillet, A. Laurent, M. Teisseire, J. Chauché
LIRMM UMR CNRS 5506
Université Montpellier II
161, Rue Ada 34392 Montpellier Cedex 5
FRANCE
jaillet,laurent,teisseire,chauche@lirmm.fr

ABSTRACT

Text categorization is a well-known task essentially based on statistical approaches using neural networks, Support Vector Machines and other machine learning algorithms. Texts are generally considered as bags of words without any order. Although these approaches have proven to be efficient, they do not provide users with comprehensive and reusable rules about their data. These rules are however very important for users in order to describe the trends from the data they have to analyze. In this framework, an association-rule based approach has been proposed by Bing Liu (CBA). In this paper, we propose to extend this approach by using sequential patterns in the SPaC method (Sequential Patterns for Classification). Taking order into account allows us to represent the succession of words through a document without complex and time-consuming representations and treatments such as those performed in natural language and grammatical methods. The original method we propose here consists in mining sequential patterns in order to build a classifier. We show on experiments that our proposition is relevant, and that it is very interesting compared to other methods. In particular, our method has better results than SVM when SVM do not perform well. Moreover, our approach is very interesting for huge volumes of data.

Keywords

text mining, categorization, sequential patterns

1. INTRODUCTION

Automatic text classification goes back at least to the 1960's [22]. But with the growing volume of available digital documents, researches on automatic classification have been extensively addressed in the past few years in order to define efficient and scalable methods [28, 33].

There are two distinct types of approaches in automatic classification: the supervised one and the unsupervised one. In the supervised classification or categorization, categories are defined by an expert while they are automatically learned in the other (also called clustering) [26, 13]. In this article we focus on text categorization or, in other words, on supervised learning algorithms.

Currently, existing classifiers are mostly based on statistical methods and Support Vector Machines. These methods are based on word frequencies such as TF-IDF (Term Frequency, Inverse Document Frequency) [26]. However, these methods do not provide understandable descriptions of the extracted knowledge. In order to cope with this problem, an association-rule based approach has first been proposed by Bing Liu (CBA) [19], which has been enhanced by [31], [18], [14] and [10], and by ARC-BC in [6] and ARC-PAN in [7]. Association rules have indeed been extensively studied and are very efficient and scalable.

All these methods consider each text as a so-called *bag of words* where no order between words is taken into account for categorization. This textual representation has proven to be useful and almost as efficient as complex representations which require time-consuming methods like syntactic analysis. It is thus interesting to investigate methods that take order into account and remain scalable.

In this paper, we propose thus to extend the CBA method by taking order into account. In our approach, the order management is performed by considering the extraction of sequential patterns instead of association rules. A sequential pattern is a set of items appearing consecutively in the database.

The originality of our approach SPaC (Sequential PAtterns for Categorization) is that we use sequential patterns for text categorization. For this purpose, two steps are considered. The first step consists in mining sequential patterns from texts. The granularity we consider is the sentence, meaning that each sentence is considered as an unordered set of items (words) whereas texts are considered as ordered sets of sentences. The second step consists in classifying documents using these sequential patterns.

In this paper, we show that sequential pattern-based classification is very efficient when SVM do not perform well. In

this framework, sequential patterns have three main advantages: first they provide understandable rules (contrary to SVM, Rocchio, naïve Bayes,...). Second they allow trend analysis, as shown in [17]. Third, they extract patterns that are more precise and informative than association rules. Note that our approach is evaluated using precision-recall merged in $F_\beta-$measure [25] and not only accuracy. These measures have indeed proven to be more relevant to compare text classification methods [29].

The paper is organized as follows. Section 2 presents the background of the problem being addressed by introducing sequential patterns and textual representations. Section 3 details existing methods dealing with text mining with "frequent patterns" and "sequential patterns". Section 4 details our method based on Sequential Patterns (SPaC). Section 5 shows that our method performs well on datasets in French and English. Finally, Section 6 summaries the paper and presents future work.

2. PROBLEM STATEMENT

First, we formulate the concept of sequence mining by summarizing the formal description of the problem introduced in [3] and extended in [30]. Second we look at the categorization problem.

2.1 Mining of sequential patterns

Let DB be a set of customers' transactions where each transaction T consists of customer-id, transaction time and a set of items involved in the transaction.

Let $I = \{i_1, i_2, ..., i_m\}$ be a set of literals called *items*. An *itemset* is a non empty set of items. A sequence s is a set of itemsets ordered according to their time stamp. It is denoted by $< s_1 s_2 ... s_p >$ where s_j, $j \in 1..n$, is an itemset. A *n-sequence* is a sequence of n items (or of length n). For example, let us consider a given customer who purchased items $1, 2, 3, 4, 5$, according to the following sequence: $s =<$ (1) (2, 3) (4) (5)$>$. This means that apart from 2 and 3 that were purchased together, i.e. during the same transaction, items in the sequence were bought separately. s is a 5-sequence.

A sequence $< s_1 s_2 ... s_p >$ is a sub-sequence of another sequence $< s'_1 s'_2 ... s'_m >$ if there exist integers $i_1 < i_2 < ...i_j... < i_n$ such that $s_1 \subseteq s'_{i_1}, s_2 \subseteq s'_{i_2}, ..., s_p \subseteq s'_{i_n}$. For example, the sequence $s' =<$ (2) (5)$>$ is a sub-sequence of s because (2) $\subseteq$ (2, 3) and (5) $\subseteq$ (5). However $<$ (2) (3) $>$ is not a sub-sequence of s since items were not bought during the same transaction.

All transactions from the same customer are grouped together and sorted in increasing order. They are called a *data sequence*. A support value (*supp(s)*) for a sequence gives its number of actual occurrences in DB. Nevertheless, a sequence in a data sequence is taken into account only once to compute the support even if several occurrences are discovered. In other words, the support of a sequence is defined as the fraction of total distinct data sequences that contain s. A data sequence contains a sequence s if s is a sub-sequence of the data sequence. In order to decide whether a sequence is frequent or not, a minimum support value (*minSupp*) is specified by the user. The sequence is said to be *frequent* if the condition $supp(s) \geq minSupp$ holds.

Given a database of customer transactions the problem of sequential pattern mining is to find all sequences whose support is greater than a specified threshold (minimum support). Each of which represents a *sequential pattern*, also called a *frequent* sequence.

Sequential patterns are usually extracted from a database built on the following schema: $date, client, items$. For instance, we consider the database of client purchases in a supermarket, as shown in Table 1. Each line (transaction, tuple) from this table corresponds to the set of items bought by the client at the corresponding date.

In this example, Peter has bought the items $1, 2, 3, 4, 5$ as the sequence $<$(1)(2,3)(4)(5)$>$.

Client	Date	Items
Peter	04/01/12	TV (1)
Martin	04/02/28	Chocolate(5)
Peter	04/03/02	DVD_Player (2) , Camera (3)
Peter	04/03/12	Printer (4)
Peter	04/04/26	Chocolate (5)

Table 1: Database of Purchases

2.2 Textual Representation and Categorization

Text categorization is the task of assigning a boolean value to each pair (document, category). A categorization process has to define: (1) a formalization of texts and category (2) a (text, category) measurement (3) a categorization policy.

The textual database is partitioned into two databases T_{Train} and T_{Test} where T_{Train} stands for a training set and T_{Test} stands for a test set.

In usual methods, texts are represented as bags of words [29]. The order is not considered. Each document is represented by a vector where each component is a word weighted by a numerical value. The most used weighting is $TF\text{-}IDF$ (Term Frequency - Inverse Document Frequency)[26]. For a word w, we have:

$$tfidf(w) = tf(w).log\frac{N}{df(w)}$$

where $tf(w)$ is the number of occurrences of w in the document, $df(w)$ if the number of documents containing w and N is the total number of documents. The weight $tfidf(w)$ represents thus the relative importance of the word in the document.

These vectors describing documents are then used in order to extract knowledge using common algorithms such as k-nearest neighbors, SVM, naive bayes, etc.

3. RELATED WORK

Text mining has been widely investigated [17, 5, 1, 29]. In this section, we focus on text classification and frequent patterns.

3.1 Classification Based on Associations: the CBA Method

In [20] the authors propose CBA: a text categorization method based on association rules. CBA consists of two parts, a rule generator (CBA-RG) which is based on the well-known Apriori algorithm [2] and a classifier builder (CBA-CB) which is based on generated rules.

3.1.1 CBA-RG

In this first step, the key is to find all ruleitems whose support is greater than a specified threshold (minimum support). A ruleitem is defined by: $< conset, C_i >$ where condset is a set of items and C_i is a class label. Each ruleitem ρ represents a rule $condset \rightarrow C_i$ and the support and the confidence of such a rule is defined by:

$$sup(\rho) = \frac{\#\text{texts from } C_i \text{ matching condset}}{|D|}$$

$$conf(\rho) = \frac{\#\text{texts in } C_i \text{ matching condset}}{\#\text{texts in D matching condset}}$$

CARs (classification rules) thus consists of all the ruleitems that satisfy the minimum support and a minimum confidence.

In these approaches, frequent patterns are extracted using a single minimum support threshold. However, categories are not always equi-distributed. It is thus not relevant to consider such a single value. Choosing a relevant minimum support threshold is crucial so that frequent patterns are relevant for the categorization task. A high support will indeed prevent the system from finding frequent patterns for a small category, while a low support will lead to the generation of a huge number of rules, which is not interesting because it will lead to overfitting.

Works have been proposed in order to define a multiple minimum support application (msCBA) [14, 21]. In these approaches, ruleitems are extracted using a multiple minimum support strategy. The minimum support of each category is defined according to the distribution frequency of each category and the user minimum support threshold:

$$minSup_{C_i} = minSup_{user} * freqDistr(C_i)$$

where, $freqDistr(C_i) = \frac{\#\text{texts from } C_i}{\#\text{texts}}$.

3.1.2 CBA-CB

Let R be the set of CARs and D the training data. The basic idea of the algorithm is to choose a set of high precedence rules in R to cover D. The categorizer is thus represented by a list of rules $r_i \in R$ ordered according a total order based on the confidence. We have thus, $< (r_1, r_2, ..., r_k), C_i >$ (where C_i is the target category and r_j one of the associated rules).

Each rule is then tested over D. If a rule does not improve the accuracy of the classifier, this rule and the following ones are discarded from the list.

Once the categorizer has been built, classification rules are tested until a condition is matched. The text is then associated with the target class of the classification rule.

3.2 Enhancements and Other Approaches

In [14], the authors propose to replace the confidence by the intensity of implication when sorting the rules to build the classifier. In [16], the authors integrate the CBA method with other methods such as decision trees.

In [10], the authors investigate a method for association rule classification. Contrary to CBA-CB which takes only one rule into account, they propose to determine the category of a text by considering several rules that are mixed using a majority voting. In order to cope with the huge number of rules, the authors propose a pruning method during the extraction of the classification rules using χ^2, as done in [18]. *maxrules* stands for the maximum number of rules used to classify new cases. Moreover, rules are used at different levels. This approach is enhanced in [9] by considering several minimum support thresholds.

In [5], association rules are used for partial classification. Partial classification means that the classifier does not cover all cases. In particular, this work is interesting when dealing with missing values.

[17, 32] propose to use sequential patterns in the text mining framework. In [32], the proposition is based on two methods. The first method is based on the visualization of word occurrences in order to detect sequential patterns. The second method is based on classical methods to extract sequential patterns. However, the authors do not propose a method to classify texts from the sequential patterns. Moreover, the texts considered are associated with a date. The corpus consists of 1,170 articles collected over 6 years. This point makes it very different from our proposal and more difficult to apply since texts are rarely associated with a date.

In [17], authors demonstrate how sequential patterns are useful for text mining. Sequential patterns are used in order to extract trends from textual databases.

As we show in this section, text mining with frequent patterns is either performed as a classification task using association rules, or as a non classification task using sequential patterns. Note that [6, 7] propose a classification method based on association rules. However this method is neither based on sequential patterns nor devoted to text classification. We propose thus an original method based on sequential patterns for classification. We argue that this method is able to deal with order in texts without being time-consuming. Next section details our approach. Section 5 shows that our method obtains good results compared to other ones.

4. SEQUENTIAL PATTERNS FOR CLASSIFICATION: THE SPAC METHOD

In this paper, we propose an original method (SPaC) for text classification based on sequential patterns. This methods consists in two steps. In the first step, we build sequential patterns from texts. In the second step, sequential patterns are used to classify texts.

4.1 From Texts to Sequential Patterns

Each text is a set of words. Our method is based on sequential pattern mining. Texts are represented as ordered sets of words using the $TF\text{-}IDF$ representation. Each text is thus considered as being the equivalent of a client. The text is constituted by a set of sentences which is associated with a date (its position in the text). Finally the set of words contained in each sentence corresponds to the set of items purchased by the client in the market basket analysis framework. Table 2 summaries the two terminologies.

Note that if order is considered in the text, a sentence remains treated as a bag of words. This adds flexibility to our approach since we argue that the order of the words within a sentence is not as important as the order of ideas in the sentences themselves.

This representation is coupled with a stemming step and a *stop-list*. The stemming step consists in replacing each word by its root word. The stop-list prevents the system from learning from noisy words such as "*the, a*".

Some words are discarded by considering the entropy of each stem over the corpus. This method eliminates words that could skew the classifier since they are not discriminant enough. Moreover, this method allows us to apply low supports in the sequential pattern discovery without deteriorating results. For this purpose, a user-defined threshold is considered. For each word w, we consider its entropy $H(w)$ over all classes C_i defined as:

$$H(w) = -\sum_{C_i} \Big[\; p(w).p(C_i|w).log(p(C_i|w)) + ((1-p(w)).p(C_i|\bar{w}).log(p(C_i|\bar{w}))) \; \Big]$$

Usual Databases		Textual Databases
client	$\leftrightarrow$	text
item	$\leftrightarrow$	word
items/transaction	$\leftrightarrow$	sentence (set of words)
date	$\leftrightarrow$	position of the sentence

Table 2: Application of the Sequential Pattern Terminology to Textual Data

In the sequel of the paper, we use the notations introduced in Table 3. Experiments have led us to consider a threshold that eliminates about 5 to 10% (according to the Zipf law [27]) of the words.

In SPaC, the sequential patterns are extracted using a multiple minimum support strategy as done in msCBA. This means that a different support is applied for each category C_i. Contrary to msCBA, the minimum supports are defined automatically by considering 3 texts per category. This assumption leads to consider for each category C_i the following multisupport:

$$minsup_{C_i} = \frac{3 * 100}{\#\text{texts in training set } C_i}$$

In our approach the training set is divided into n training sets, one for each category. Texts are thus grouped depending on their category. Sequential pattern mining algorithms are applied separately on these n databases using the corresponding minimum supports.

For each category, frequent sequential patterns are computed and their supports are stored. The support of a frequent pattern is the number of texts containing the sequence of words.

Notation	Meaning
$\mathcal{C} = \{C_1, \ldots, C_n\}$	set of n categories.
$C_i \in \mathcal{C}$	a given category.
$minSup_{C_i}$	user-defined minimum support for category C_i.
T	set of texts.
$T^{C_i} \subseteq T$	set of texts belonging to category C_i.
$T_{Train} = \{(C_i, T^{C_i})\}$	Training set constituted by a set of texts associated with their category.
SEQ	set of sequences found for category C_i, customer c at time t.
SP	table of sequential patterns.
$RuleSP$	table of tuples $(sp_j, C_i, conf_{i,j})$. corresponding to the sequence sp_j, the category C_i and the confidence $conf_{i,j}$ of the rule $sp_j \rightarrow C_i$.

Table 3: Notations

DEFINITION 1. *Let $< s_1 \ldots s_p >$ be a sequence. The support of $< s_1 \ldots s_p >$ is defined as:*

$$supp(< s_1 \ldots s_p >) = \frac{\#texts\ matching < s_1 \ldots s_p >}{\#texts}$$

Algorithm 1 describes the SPaC sequential pattern generation. The SPAM algorithm is used through the $SPMining()$ function in order to find all frequent sequences in the transactional databases (DB) [8].

Algorithm 1: SPaC rules generation

Data : T_{Train} : the training Set
$\{minSup_{C_i}\}$: the set of minimum supports for each category C_i

Result: SP: the set of sequential patterns

begin
  $SEQ \leftarrow \emptyset$; $cust \leftarrow 0$; $date \leftarrow 0$;
  foreach *Category* $C_i \in \mathcal{C}$ **do**
    foreach *Text* $T_j \in T^{C_i}$ **do**
      foreach *Sentence* $S_k \in T_j$ **do**
        V_s=TFIDF(Stem(S_k)); // Compute the $TF\text{-}IDF$ vector of the sentence
        for $(s = 0; s <\mid V_s \mid; s++)$ **do**
          if $V_s[s] > 0$ **then**
            $SEQ[C_i][cust][date]$.additem(s);
        date++;
      $date \leftarrow 0; cust++$;
    $cust \leftarrow 0$;
  foreach *Category* $C_i \in C$ **do**
    SP[C_i]=SPMining(SEQ[C_i],$minSup_{C_i}$);
end

For instance, the following frequent patterns have been extracted from the category "Purchasing-Logistics" of our French database:

< (cacao) (ivoir) (abidjan)>
< (blé soja) (maï)>
< (soj)(blé lespin victor)(maï soj)(maï)(grain soj)(soj)>

The first sequential pattern means that texts contain words *cacao* then *ivoire* then *Abidjan* in three different sentences. The second sequential pattern means that texts contain the words *blé* and *soja* in the same sentence and then *maïs*[1] (*mai*). The third sequential pattern means the word *maï* occurs in two successive sentences before the word *grain*.

Note that the use of sequential patterns allows the apparition of a word several times in the text, contrary to association rules. Moreover, some frequent co-occurrences can be identified with sequential patterns mining.

4.2 From Sequential Patterns to Categories

Once sequential patterns have been extracted, the goal is to derive a categorizer from the obtained rules. This is done by computing for each rule in a category the corresponding *confidence*. For each sequential pattern $< s_1 \dots s_p >$ found for the category C_i, a rule γ is computed:

$\gamma :< s_1 \dots s_p > \rightarrow C_i$.

This rule means that if a text contains s_1 then $s_2 \dots$ then s_p then it will belong to the category C_i. Each rule is associated with its confidence, indicating to which extent the sequential pattern is characteristic to this category:

[1]In French, *blé* means *wheat* and *maïs* stands for *corn*

$$conf(\gamma) = \frac{\#\text{texts from } C_i \text{ matching} <s_1 \dots s_p>}{\#\text{texts matching} <s_1 \dots s_p>}$$

Algorithm 2: SPaC-C Method

Data : T_{Test}: the test Set
KFS: the K First Satisfied Parameters
SP: the sequential Pattern Table from SPaC-RG

begin
  $nb \leftarrow 1$;
  foreach *Category* $C_i \in C$ **do**
    foreach $sp_j \in SP[C_i]$ **do**
      $RuleSP[nb] \leftarrow (sp_j, C_i, conf(sp_j \rightarrow C_i)$;
      $nb++$;
  Sort $RuleSP$ by rule confidence and size of sequence ;
  $nfs \leftarrow 0$; $classable \leftarrow 0$;
  foreach *Text* $T_k \in T_{Test}$ **do**
    foreach *rule* $(sp_j \rightarrow C_i) \in RuleSP$ **do**
      if T_k *supports* SP_j **then**
        T_k.score[C_i]++; $classable \leftarrow 1$; nfs++ ;
        if $nfs \geq KFS$ **then** break
    if *classable* **then**
      Set T_k to the Most Valued Category;
    classable $\leftarrow 0$; nfs $\leftarrow 0$;
end

Rules are sorted depending on their confidence and the size of the associated sequence. When considering a new text to be classified, a simple categorization policy is applied: the K rules having the best confidence and being supported are applied. The text is then assigned to the class mainly obtained within the K rules. This method is the same as the majority voting in [10]. If two categories obtain the same score, a random choice is carried out. This prevents the system from always choosing the same category. The SPaC classifier step (SPaC-C) is described in Algorithm 2.

5. EXPERIMENTS

Experiments are led on three databases. The two first ones are the well-known english databases 20 Newsgroups and Reuters [12]. The third database describes French texts (news) with 8239 texts divided into 28 categories.

Experiments compare our approach to the results obtained with CBA and SVM. Table 4, 5 and 6 detail these results. Comparisons are based on the F_β measure [29]. This measure allows us to combine recall and precision for a global evaluation. The F_β measure is thus more relevant than accuracy since accuracy does not take into account the case when a text is not classified. This leads to consider that classifying no text (thus never making errors) would have an accuracy near 100%! Accuracy has been taken as the reference measure in [10, 19]. However, for the reasons presented above, we argue that this is not as relevant as relying on recall and precision, as mentioned in [29]

Definition 2. *The F_β measure is defined as follows:*

$$F_\beta = \frac{(\beta^2 + 1)\pi_i \rho_i}{\beta^2 \pi_i + \rho_i}$$

where ρ stands for the recall, and π stands for the precision. This measure is computed for each class C_i.

DEFINITION 3. *Precision and recall are defined as follows:*

$$\pi_i = \frac{TP_i}{TP_i + FP_i} \quad , \quad \rho_i = \frac{TP_i}{TP_i + FN_i}$$

where TP_i, FP_i, FN_i stand respectively for the number of texts in class well classified i (True Positive), the number of texts put in class C_i by error (False Positive), the number of texts put in a different class from i by error (False Negative).

Accuracy is defined as follows:

DEFINITION 4. *Accuracy:*

$$Accuracy_i = \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}$$

In order to evaluate the classifier for all classes, we consider *Micro-averaging* (μ) and *Macro-averaging* (M) [29, 34] defined as follows:

DEFINITION 5. *Macro-averaging and Micro-averaging*

$$\hat{\pi}^M = \frac{\sum_{i=1}^{|C|} \hat{\pi}_i}{|C|}, \hat{\rho}^M = \frac{\sum_{i=1}^{|C|} \hat{\rho}_i}{|C|}$$

$$\hat{\pi}^\mu = \frac{\sum_{i=1}^{|C|} VP_i}{\sum_{i=1}^{|C|}(VP_i + FP_i)}, \hat{\rho}^\mu = \frac{\sum_{i=1}^{|C|} VP_i}{\sum_{i=1}^{|C|}(VP_i + FN_i)}$$

Micro-averaging gives the same importance to each document, contrary to *macro-averaging* which computes the average class by class (puting thus more importance to small categories).

In this paper, we consider that recall and precision have the same importance. We have thus: $\beta = 1$ for the F_β measure.

	SPaC	msCBA	SVM
$F1^M$	0.444	0.367	**0.485**
$F1^\mu$	**0.487**	0.401	0.486
Accuracy	0.962	0.956	**0.969**
Parameters	multisupp.	multisupp. = 0.7%	C = 1

Table 4: Comparison of results by SPaC, msCBA and SVM on French news. Training Set = 33%

Results for SVM are obtained using a linear kernel (no better result is provided by a more complex kernel) with SVMLight [15]. The supports chosen here are the values providing the best results while remaining computable (too low supports lead to a very time-consuming application). SPaC is applied with $K = 10$ rules taken into account when classifying new documents.

In this work, we argue that getting understandable knowledge is as important, and even more important than getting the most accurate classifier. For this reason, comparisons are essentially studied between CBA and SPaC. CBA is tested with the msCBA version of the algorithm, using the best results we have obtained when testing different supports. The results show that SPaC is always better than CBA when considering the macro-average. This is due to the fact that SPaC obtains good results for every category while CBA obtains very good results for the categories having many texts and low results for little categories. SPaC is thus more efficient when dealing with a difficult learning task. For this reason, SPaC is better than CBA and similar to SVM for French texts, and is better than SVM when dealing with the 20 Newsgroups database.

	SPaC	msCBA	SVM
$F1^M$	0.219	0.082	**0.500**
$F1^\mu$	0.591	0.679	**0.840**
Accuracy	0.990	0.992	**0.996**
Parameters	multisupp.	multisupp. = 1%	C = 1

Table 5: Comparison of results by SPaC, msCBA and SVM on English texts (REUTERS)

	SPaC	msCBA	SVM
$F1^M$	**0.463**	0.423	0.423
$F1^\mu$	**0.502**	0.436	0.455
Accuracy	**0.947**	0.941	0.941
Parameters	multisupp.	multisupp. = 3.5%	C=1

Table 6: Comparison of results by SPaC, msCBA and SVM on English texts (20 Newsgroups). Training Set = 33%

In Fig. 1 we compare the results obtained for the $F1$ measure regarding the number of rules considered for the choice of the class. Experiments have shown that $K = 10$ provides good results (similar to [10] where best results correspond to $maxrules = 9$). Experiments are done (1) with order: the sequential pattern (sequence) is supported by the text (2) without order: the sub-sequences are supported by the text in an unspecified order. The corresponding sentences are in the document but they are not ordered as in the sequential pattern. We can notice that results are always better when order is taken into account.

6. CONCLUSION AND FURTHER WORK

In this paper, we address the problem of text categorization using sequential patterns. In our framework, texts are represented by $TF\text{-}IDF$ vectors, and each category is associated with a set of sequential patterns. When classifying new data, a text is matched to a category depending on the number of sequential patterns holding. The corresponding category is determined using a majority voting. Even if SVM have proven to be efficient in such a task, we argue that it is very important to provide users with understandable knowledge about their data. In this framework, sequential patterns

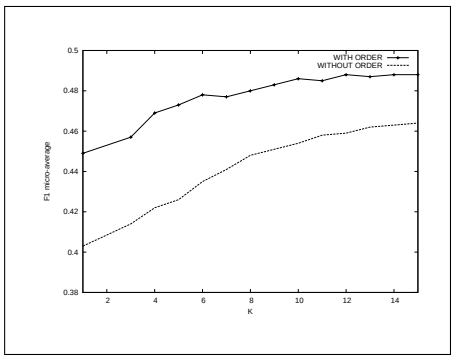


Figure 1: SPaC: $F1$ in function of the number of rules considered

are well-adapted. They provide rules that are used for the classification. We show that this approach is efficient and relevant, in particular when SVM do not perform well.

Moreover, the method we propose is simple and adaptable to drifting concepts since it is possible to update sequential patterns without performing the whole process using incremental sequential patterns mining [23]. This possibility is of high importance for text categorization, in particular for the automatic analysis of news which is a very fast variable area. This is thus a first step towards On-Line Classification Process (OLCP) using sequential patterns.

Future works include the integration of our approach on different foreign languages, in order to determine how order is important for each language. We are working on the automatic definition of the best number number of rules to take into account (the K parameter of our method). Our approach may also be enhanced by mining generalized sequential patterns [4]. This framework allows to integrate time constraints as shown in [24]. Finally, we aim at integrating muti-level sequential patterns, as proposed in [9]. This allows indeed to keep very specific rules without damaging the classifier performances. In this framework, we argue that it is interesting to build a very compact set of rules (rules where the left-hand part is as short as possible). For this purpose, we aim at extending works on $\delta-$free sets [11] to sequential patterns.

7. REFERENCES

[1] R. Agrawal, R. J. B. Jr., and R. Srikant. Athena: Mining-based interactive management of text databases. In *Extending Database Technology*, pages 365–379, 2000.

[2] R. Agrawal and R. Srikant. Fast Algorithms for Mining Generalized Association Rules. In *Proc. of the 20th Int. Conf. on Very Large Databases (VLDB'94)*, September 1994.

[3] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proc. of the 11th Int. Conf. on Data Engineering (ICDE'95)*, Tapei, Taiwan, March 1995.

[4] R. Agrawal and R. Srikant. Mining sequential patterns. In *Eleventh Int. Conf. on Data Engineering*, pages 3–14. IEEE Computer Society Press, 1995.

[5] K. Ali, S. Manganaris, and R. Srikant. Partial classification using association rules. In *Knowledge Discovery and Data Mining*, pages 115–118, 1997.

[6] M.-L. Antonie and O. Zaiane. Text document categorization by term association. In *IEEE International Conference on Data Mining (ICDM'2002)*, pages 19–26, 2002.

[7] M.-L. Antonie and O. Zaiane. An associative classifier based on positive and negative rules. In *9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD-04)*, pages 64–69, 2004.

[8] J. Ayres, J. Gehrke, T. Yiu, and J. Flannick. Sequential pattern mining using bitmaps. In *Proc. of the 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining.*, 2002.

[9] E. Baralis, S. Chiusano, and P. Garza. On support thresholds in associative classification. In *Proc. of the 2004 ACM Symposium on Applied Computing (SAC)*, pages 553–558, 2004.

[10] E. Baralis and P. Garza. Majority classification by means of association rules. In *7th European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 35–46, 2003.

[11] B. Cremilleux and J. Boulicaut. Simplest rules characterizing classes generated by delta-free sets. In *Proceedings of the 22nd BCS SGAI International Conference on Knowledge Based Systems and Applied Artificial Intelligence ES 2002*, pages 33–46. Springer-Verlag, 2002.

[12] S. Hettich and S.D.Bay. *The UCI KDD Archive. [http://kdd.ics.uci.edu]*. Irvine, CA: University of California, Department of Information and Computer Science., 1999.

[13] M. Iwayama and T. Tokunaga. Cluster-based text categorization: a comparison of category search strategies. In *Proc. of SIGIR-95, 18th ACM Int. Conf. on Research and Development in Information Retrieval*, pages 273–281. ACM Press, 1995.

[14] D. Janssens, G. Wets, T. Brijs, K. Vanhoof, and C. G. Adapting the cba-algorithm by means of intensity of implication. In *Proc. of the First Int. Conf. on Fuzzy Information Processing Theories and Applications*, pages 397–403, 2003.

[15] T. Joachims. Text categorization with support vector machines: learning with many relevant features. In *Proc. of ECML-98, 10th European Conf. on Machine Learning*, pages 137–142, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.

[16] V. Kumar and et al, editors. *Classification Using Association Rules: Weaknesses and Enhancements*, 2001.

[17] B. Lent, R. Agrawal, and R. Srikant. Discovering trends in text databases. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining, KDD*, pages 227–230. AAAI Press, 14–17 1997.

[18] W. Li, J. Han, and J. Pei. CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules. In *Proc. 2001 Int. Conf. on Data Mining (ICDM'01)*, 2001.

[19] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Knowledge Discovery and Data Mining*, pages 80–86, 1998.

[20] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Knowledge Discovery and Data Mining*, pages 80–86, 1998.

[21] B. Liu, Y. Ma, and C. K. Wong. Improving an association rule based classifier. In *Principles of Data Mining and Knowledge Discovery*, pages 504–509, 2000.

[22] M. Maron. Automatic indexing: An experimental inquiry. *Journal of the ACM (JACM)*, 8:404–417, 1961.

[23] F. Masseglia, P. Poncelet, and M. Teisseire. Incremental mining of sequential patterns in large databases. *Data and Knowledge Engineering*, 46(1), 2003.

[24] F. Masseglia, P. Poncelet, and M. Teisseire. Pre-processing Time Constraints for Efficiently Mining Generalized Sequential Patterns. In *11th Int. Symposium on Temporal Representation and Reasoning, TIME'04*, July 2004.

[25] C. J. V. Rijsbergen. *Information Retrieval*. Butterworths, sec. edition, 1979.

[26] G. Salton and M. J. McGill. *Introduction to modern information retrieval*. 1983.

[27] G. Salton, C. Yang, and C. Yu. A theory of term importance in automatic text analysis. *Journal of the American Society for Information Science*, 36:33–44, 1975.

[28] F. Sebastiani. Machine learning in automated text categorisation. In *Proc. of ACM Computing Surveys*, volume 34, pages 1–47, 2002.

[29] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[30] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proc. of the 5th Int.Conf. on Extending Database Technology (EDBT'96)*, pages 3–17, September 1996.

[31] K. Wang, S. Zhou, and Y. He. Growing decision trees on support-less association rules. In *Knowledge Discovery and Data Mining*, pages 265–269, 2000.

[32] P. C. Wong, W. Cowley, H. Foote, E. Jurrus, and J. Thomas. Visualizing sequential patterns for text mining. In *INFOVIS*, pages 105–114, 2000.

[33] Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1/2):69–90, 1999.

[34] Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval Journal*, 1:69–90, 1999.

Finding Complex Patterns over Data Streams

Leila Kaghazian Dennis McLeod Reza Sadri
Integrated Media Center Systems *Procom Technology*
University of Southern California *Irvine,CA*
{kaghazia,mcleod}@usc.edu *sadri@procom.com*

ABSTRACT

.
Searching data streams has been traditionally very limited, either in the complexity of the search or in the size of the searched dataset. We present a framework for searching complex patterns in very large data streams, but introducing simple extensions to SQL along with and optimization method for complex sequential searches. Our experimental results demonstrate impressive speedup.

1. INTRODUCTION

Recently, a significant number of applications require the process of data streams. Examples include online stock analysis, computer network monitoring, network traffic management, telecommunication data, transactions and retail chains, auction market data and earthquake prediction. The major common characteristic of the above applications is that they are all time critical and they are in stream format. Therefore, the DBMS must be equipped by effective and efficient tools for data stream processing. Sometimes, the only feasible way to make sense of large volumes of data is to exact search for patterns of interest. This is especially difficult when the patterns of interest are complex. Traditional constructs available in SQL can't express these rich patterns. Facilities like datablades have increased the expressive power of database query languages, but still there are applications that need a more expressive language for describing their patterns of interest. Another limitation of most of these applications is that data is processed on the fly and there is a limited buffer for keeping the history of the time-series; therefore, we are in need of an implementation of the pattern detection mechanism that isn't bound to keeping the whole history of the sequence.

Looking to extend SQL with the ability to query the time-series data bases with more flexibility and power than Informix datablades [11] and SRQL [16] , Sadri et al in [18] introduced an extension of SQL, SQL-TS, to express sequential patterns, and studied how to optimize search queries for this language. They exploit the inter-dependencies between the elements of a sequential pattern to minimize repeated passes over the same data. While proposed technique in [18] is powerful enough to find many types of patterns, it lacks the power necessary for expressing some interesting queries. For instance, it has not designed to search for patterns including nested stars (recurring pattern inside another recurring pattern). In this paper we extend such algorithm as Optimal Pattern Search (OPS*) and present a general algorithm which gives the SQL-TS the capability to look for more complex patterns than its previous versions including nested-star patterns. OPS* provides a general framework to search for any context free pattern in SQL level. Our preliminary result is encouraging and it shows more expressive power plus an impressive speedup. At the end we illustrate ways to improve the algorithm and determine the requirements for extending it for searching complex patterns specifically in the stream data.

2. RELATED WORKS

Sequential pattern search is an important problem with broad applications. A major group of research in this area has focused on discovering frequent patterns in sequential data (such as time series). The main focus of these works is to discover frequent patterns through approximation, transformation [1] and statistical inference. For works in Artificial Intelligence look at [12] [23] [8] and in the database context, where input data is usually much larger, the problem has been studied in a number of recent papers [2, 3, 8, 14, 25]. We are not looking for frequent patterns in time series; rather we are interested in exact match of a pattern in the SQL-TS level.

Another important work related to sequential pattern is the extension of database query languages with the ability of searching for and manipulating sequential patterns. This includes Informix extension to SQL[11], SEQUIN [21, 22, 20]. SRQL [16], CQL [24] and SQL/LPP [15]. Our work is independent of these researches. While we use SQL-TS as an alternative to these approaches, the main issue addressed in this paper is introducing a context free algorithm to search for complex pattern and could be employed by all extended version of SQL for time series.
During the last years, data streams have attracted the interest of many researchers. STREAM [4, 26] is a data stream processing project whose focus is on computing approximate results and to understand how to efficiently run queries in a bounded amount of memory. The Aurora

[5] system allows users to specify quality-of-service requirements for queries, and then uses those specifications to determine how and when to shed load. Other recent research has focused on developing algorithms to perform specific functions on sequenced data. Gehrke et al. [9] considers the problem of computing correlated aggregate queries over streams, and presents techniques for obtaining approximate answers in a single pass. Yang et al. [27, 28] discusses data structures for computing and maintaining aggregates over streams. Sadri et al. [18] propose SQL-TS, an extension of the SQL language to express sequence queries over time-series data. Finally, there has been a spate of work on this topic more recently, especially from the group at IIT-Bombay [7, 10, 17, 19]. Multi-query optimization typically shares relational sub expressions that appear in the plans of multiple (snapshot) queries. The Telegraph and TelegraphCQ [6] project have developed a suite of novel technologies for continuously adaptive query processing and on meeting the challenges that arise in handling large streams of continuous queries over high-volume, highly-variable data streams. Our proposed OPS* algorithm is a pattern detection mechanism that isn't bound to keeping the whole history of the sequence and trying to optimize the search by exploiting the inter-dependencies between the elements of a sequential pattern to minimize repeated passes over the same data.

3. SQL-TS

Simple Query Language for Time Series (SQL-TS), introduced in Sadri et all [18], adds simple constructs to SQL for specifying complex sequential patterns. SQL-TS is identical to SQL, except for the following additions to the FROM clause:

- A SEQUENCE BY clause specifying the sequencing attributes.
- A CLUSTER BY clause specifying the grouping attributes, similar to GROUP BY. Each group indicates a separate sequence.

To clarify the subject, suppose that we have a stream containing the bids of ongoing auctions, as follows:

auction_id : *id for specific auctioned item*
amount : *amount of bid*
time : *Timestamps*

To find the three consecutive bids with more than %20 increase, we can write the SQL-TS query of Example 1:

Example 1. finding the three consecutive bids with more than %20 increase.

```
SELECT   T.auction_id,T.amount,T.tim
FROM bids
        CLUSTER BY auction_id
        SEQUENCE BY time
        AS (X,Y,Z,T)
WHERE 1.2*X.amount< Y.amount
        AND 1.2* Y.amount  <Z.amount
        AND 1.2*Z.amount < T.amount
```

The AS clause, which in SQL is mostly used to assign aliases to the table names, is used to specify a sequence of tuple variables from the specified table. Tuple variables from this sequence can be used in the WHERE clause to specify the conditions for expressing the pattern, and in the SELECT clause to specify the output.

A key feature of SQL-TS is its ability to express recurring patterns by using a star operator. However, star operator can be applied only to simple patterns, and not to complex patterns that contain sub-patterns. We extend the support for recurring complex patterns, as it is described in section 4.

4. OPS

By extending the KMP text matching algorithm [13], Sadri et al [18], introduced Optimized Pattern Search (OPS), in order to optimize sequential queries in SQL-TS. Following is a brief summary of OPS:

Given an input stream and a sequential query, suppose that while searching for the sequential pattern on an input stream, a mismatch occurs at the $j\,th$ position of the pattern. Speedup is achieved by tracking two items, *shift(j)* and *next(j),* that help resetting the position trackers (i and j) to optimize values after the mismatch . *Shift(j)* determines how far the pattern should be advanced in the input, and *next(j)* determines from which element in the pattern the checking of conditions should be resumed after the shift. To compute *shift(j)* and *next(j)*, OPS algorithm begins by capturing all the logical relations among pairs of the pattern elements using a positive precondition logic matrix θ, and a negative precondition logic matrix ϕ. These matrices are of size m, where m is the length of the search pattern. The θ_{jk} and ϕ_{jk} elements of these matrices are only defined for $j \geq k$; thus there is a lower-triangular matrices of size m. θ_{jk} and ϕ_{jk} are defined as follow:

$$\theta_{jk} = \begin{cases} 1 & if \quad p_j \Rightarrow p_k \wedge p_j \neq F \\ 0 & if \quad p_j \Rightarrow \sim p_k \\ U & otherwise \end{cases}$$

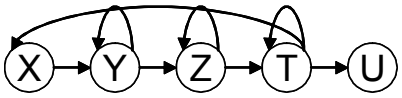


Figure 1: State model for Example 2

$$\phi_{jk} = \begin{cases} 1 & if \quad \sim p_j \Rightarrow p_k \\ 0 & if \quad \sim p_j \Rightarrow \sim p_k \wedge p_j \neq T \\ U & otherwise \end{cases}$$

in which p_i is the predicate at location i. From matrices θ and ϕ, matrix S is derived that describes the logical relationships between whole patterns. *next* an *shift* are consequently derived using matrix S.

5. PATTERNS WITH NESTED STAR

An important advantage of the OPS algorithm is that it can be easily generalized to handle input patterns which, in SQL-TS, are expressed using the star. Consider the following generalized predicates:

$$p_1(t) = t_i.price < t_{i-1}.price$$
$$p_2(t) = t_i.price > t_{i-1}.price$$

then $*(*p_1, *p_2)$ matches the sequences of records with recurring patterns of decreasing prices following by a period of increasing prices. For instance take the following SQL-TS example:

Assume we have a stream of earthquake data containing the magnitude of quakes in predefined regions:

latitude : *latitude of the region*
longitude : *longitude of the region*
region : *region name*
magnitude: *magnitude of a quake*
time: *Timestamp*

Example 2: Suppose that we are looking for repeated occurrences of the following pattern in quake magnitude, leading to a quake with magnitude over 3.5. The pattern starts with a quake of magnitude between 1.8 and 3.0 followed by a series of quakes with decreasing magnitude, followed by a series of quakes with magnitudes of almost the same rate (within 0.1 Richter), followed by a series of quakes with increasing magnitude. The query written in SQL-TS is:

```
SELECT  X.first.magnitude, X.first.time,
        U.previous.magnitude, U.previous.time
FROM earthquake
        CLUSTER BY region
        SEQUENCE BY time
        AS (*(X,*Y,*Z,*T),U)
WHERE
        X.region="Los Angeles"
        AND 1.8<X.magnitude
        AND X.magnitude<2.5
        AND Y. magnitude <Y.previous. magnitude
        AND 0.99.*Z.previous.magnitude<
                                Z. magnitude
        AND Z.magnitude <
                        1.1*Z.previous. magnitude
        AND T.magnitude >
                        1.1*T.previous. magnitude
        AND U.magnitude > 3.5
```

Therefore our pattern predicates (on input tuple t) are:

```
p1(t) = (1.8< t. magnitude <2.5)
p2(t)=(t. magnitude < t.previous. magnitude)
p3(t)=(0.99*t.previous.magnitude  <t.magnitude  <
        1.1* t.previous. magnitude)
p4(t)=(t.magnitude > 1.1*t.previous. magnitude)
p5(t) = ( t. magnitude >3.5)
```

The calculation of logic matrices θ and ϕ, remains unchanged in the presence of nested stars patterns; thus, the formulas given in Section 3 will still be used. However, the calculation of the arrays next and shift must be generalized for nested star patterns as described next.

At runtime we maintain an array of counters (one per pattern element) to keep track of the cumulative number of input objects that have matched the pattern sequence so far. For example, if the first pattern element is a star that matched five elements in the input and the second pattern element is a non star, matching only one input element and the third element is a star matching two input elements we will have $count_1 = 5, count_2 = 6$, and $count_3 = 8$.

5.1 Proposed Algorithm for Patterns with Nested Stars

As it described, some of the counters have more than one cumulative value. Following in the algorithm for a pattern with nested stars, we will employ these values.
We represent a pattern with a finite state model in which elements of the pattern are the states of the model. Stars and nested starts are coded in state transitions. Figure 1 illustrates a state diagram for Example 2. To develop the OPS* algorithm, the next state will be creating an adjacency matrix based on the state model of the pattern. The following adjacency matrix, presents the state model of Example 2:

OPS* Algorithm:

$OPS^*(i, j)$

$i \leftarrow 1;\ j \leftarrow 1$

WHILE $((j \le m)\ and\ (i \le n))$

/* m is the length of the pattern and n is the length of the input data */

$R_j = \{k |\ \exists k\ s.t.\ j \le k\ \&\ A(j,k) == 1\}$

/* R_j presents all possible nested star element of a sub-patterns start at k and end at j */

IF the current input element satisfies the pattern,

THEN

if R_j is empty

$j \leftarrow j+1,\ i \leftarrow i+1$

else

$j \leftarrow \max(R_j),\ i \leftarrow i+1$

OTHERWISE (i.e. when the current input element doesn't satisfy the pattern)

If R_j is empty or ($j == \max(R_j)$ and p_j is tested for the first time) then

- reset j (the index in the pattern) to *next(j)* and
- reset i (the index in the input) as follows:

$$i = i - j + \min\ (count\ (shift\ (j) + next\ (j) - 1)$$

If R_j is not empty and $j == \max(R_j)$ and $R_j - \max(R_j)$ is empty then

- $j \leftarrow j+1$

If R_j is not empty and $j == \max(R_j)$ and $R_j - \max(R_j)$ is not empty then

- $j \leftarrow \max(R_j - \max(R_j))$

Figure 2 : OPS* Algorithm

$$A=\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In the above matrix, the elements of patterns (X,Y,Z,T,U) were represented as (1,2,3,4,5) respectively. Hence $A(2,2)=1$ means p_2 (the second element of the pattern) is a star element. If $A(j,k)=1$ and $k < j$,then p_j is the last element of a nested star sub-pattern. In the OPS* algorithm, for each row j, we define R_j which includes every $k, (k \le j)$ such that $A(j,k)=1$. For instance, in Example 2, $R_4=\{1,4\}$.

In Figure 2 we describe the OPS* algorithm in detail.

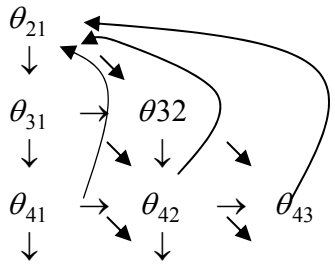
The difference between state models in OPS and OPS* is that OPS* model may include right to left transitions, however in OPS state model we only face left to right or self loop transitions. Note that in both OPS and OPS*, left to right transitions are only between adjacent states.

To complete the OPS algorithm, we must now specify the computation of *shift(j)* and *next(j)* in the presence of nested stars

.

6. FINDING *SHIFT* AND *NEXT* FOR THE NESTED STAR CASE

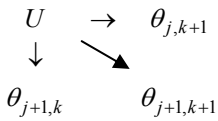
Consider the following sample graph based on the matrix θ (excluded the main diagonal):



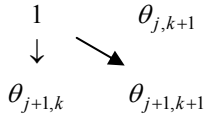
The entry θ_{jk} in our matrix correlates pattern predicates p_j with p_k, k < j, when these are evaluated on the same input element. Therefore, we can picture the simultaneous processing of the input on the original pattern, and on the same pattern shifted back by $j-k$. Thus the arcs between nodes in our matrix above show the combined transitions in the original pattern and in the shifted pattern. In particular, consider θ_{jk} where neither p_k nor p_j are star predicates; then after success in p_j and p_k, we have a transition to p_{j+1} in the original pattern, and to p_{k+1} in the shifted pattern: this transition is represented by an arc $\theta_{jk} \rightarrow \theta_{j+1,k+1}$. However, if p_j is not a star predicate, while p_k is, then the success of both will move p_k to p_{k+1}, but leave p_j unchanged: this is represented by the arc $\theta_{jk} \rightarrow \theta_{j,k+1}$. In the nested star situation, there is another possible arc which is a back edge when the last element of the nested star sub-pattern satisfies the previous input element but not the current one. In this case, before going forward to match the input element with the next pattern element, algorithm evaluates the input element against the first element of the nested star sub-pattern, so the graph will have a back edge.

In general, it is clear that only some of the arcs listed in the matrix above represent valid transitions and should be considered, the set of valid transitions also depends on the values of θ. In particular, since all the predicates in the pattern must be satisfied by the shifted input, every $\theta_{jk} = 0$ entry must removed with all its incoming and departing arcs: we only retain entries that are either *1* or *U*. Considering all possible situations, and assuming that all the neighbors are non-zero entries, we conclude that only the following transitions are needed when building the graph:

1. If both elements j and k of the pattern sequence are star predicates and $\theta_{jk} = U$, then we have three outgoing arcs from θ_{jk}: one to $\theta_{j+1,k}$, one to $\theta_{j+1,k+1}$ and one to $\theta_{j,k+1}$. Pictorially,



2. If both element j and element k of the pattern are stars and θ_{jk} = 1, we have two outgoing arcs from θ_{jk}: one to $\theta_{j+1,k+1}$ and the other to $\theta_{j+1,k}$. Pictorially,

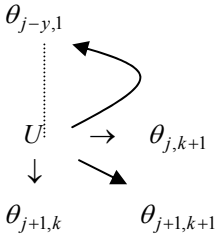


There is no arc to θ_{jk+1}, because θ_{jk} = 1; thus all input tuples that satisfy p_j must also satisfy p_k.

3. If both elements *j* and k of the pattern are non-star predicates, then we have only one arc from θ_{jk} to $\theta_{j+1,k+1}$.
4. If element *j* of the pattern is a star predicate, but element k is not, then we have two arcs from θ_{jk}: one to $\theta_{j+1,k+1}$ and the other to $\theta_{j,k+1}$.
5. If element k of the pattern is a star predicate but element j is not, then we have two arcs from θ_{jk}: one to $\theta_{j+1,k+1}$ and the other to $\theta_{j+1,k}$..

In presence of nested star, when element *j* is the last element of the nested star sub-pattern and is a star predicate, the following transitions would be added to the graph:

6. If element k of the pattern sequence is a star predicate and $\theta_{jk} = U$, then we have four outgoing arcs from θ_{jk}: one to $\theta_{j+1,k}$, one to $\theta_{j+1,k+1}$,one to $\theta_{j,k+1}$ and one to $\theta_{j-y,1}$ (where y = length (nested star sub-pattern)-1). Pictorially,



7. If element k of the pattern is a star and $\theta_{jk} = 1$, we have three outgoing arcs from θ_{jk}: one to $\theta_{j+1,k+1}$, one to $\theta_{j+1,k}$ and the other to $\theta_{j-y,1}$.
8. If element k of the pattern is not a star predicate, then we have three arcs from θ_{jk}: one to $\theta_{j+1,k+1}$, one to $\theta_{j,k+1}$ and the other to $\theta_{j-y,1}$.

These rules assume that the end nodes of the arcs have value U or 1; but when such nodes have value 0, the incoming arcs will be dropped.

The directed graph produced by this construction will be called the Implication Graph for pattern sequence P, and is denoted as G_p . For each value of j this graph must be further modified with entries from ϕ to account for the fact that j th element of the pattern failed on the input.

Therefore, we replace the j th row of G_p (i.e., the row that starts with $\theta_{j,1}$) with the j th row of matrix ϕ, and remove all rows and arcs after j. In addition we recompute the arcs from row $j-1$ to row j according to the new values of elements in row j.

Thus, if element k is star, there are up to two arcs from $\theta_{j-1,1}$ to row j: one to ϕ_{jk} and one to $\phi_{j,k+1}$. If element k is not an star, then there will be only an arc from $\theta_{j-1,k}$ to row j that goes to ϕ_{jk}. Furthermore, all the original G_p entries in rows up to and including $j-1$ remain unchanged, and so are all arcs leading to entries in these rows.

Again we assume that the end nodes of the arcs are either U or 1; but when such nodes are 0 the incoming arcs will be dropped. The resulting graph will be called the Implication Graph for pattern element j, denoted G_p^j ; this graph will be used to compute *shift(j)* and *next(j).*

6.1 Computation of *shift(j)* and *next(j)* from G_p^j

In general we define *shift(j)* as follows:

Shift. Let P denote the search pattern, and let $\sigma(j) = \{s \mid \exists$ a path from $\theta_{s+1,1}$ to a node in the last row of $G_p^j\}$.

Then

- if the set $\sigma(j)$ is not empty, then $shift(j) = \min(\sigma(j))$
- if the set $\sigma(j)$ is empty and $\phi_{j1} \neq 0$ then *shift(j)=j-1*
- if the set $\sigma(j)$ is empty and $\phi_{j1} = 0$ then *shift(j) =j.*

Next. Multiple paths leading to the last row were acceptable for *shift*, but they are not acceptable for *next*, since this must return a value that uniquely determines the point from which the search must be resumed. Therefore, let us say that a node in our G_P^j graph is *deterministic* if there is exactly one arc leaving this node, and the end-node of this arc has value 1 (thus a deterministic node cannot take us to an U node or to several 1 nodes). Thus, we start from $\theta_{shift(j)+1,1}$, and if this is not deterministic, then we set $next(j) = 1$. Otherwise, we move to the unique successor of this deterministic node and repeat the test. When the first non-deterministic node is found in this recursive process, $next(j)$ is set to the value of its column. If the search takes us to the last row in G_P^j , that means that none of the input elements previously visited needs to be tested again: thus $next(j) = j - shift(j)$.

Despite the fact that Implication Graph for the nested star algorithm may have some back edges, the computation for the *shift(j)* and *next(j)* is based on the same formula as the star algorithm. Suppose that there is a path from $\theta_{j-y,1}$ to the last row of the G_P^j. Also assume that there is a back edge from $\theta_{j-2,1}$ to the $\theta_{j-y,1}$ and there is a path from $\theta_{j-2,1}$ to the last row. Thus

$$\sigma(j) = \{j-y, j-2\}(y > 2)$$

and

$$Shift(j)=min(\sigma(j))= j-y .$$

So existence of back edge in the Implication Graph does not have any impact on the calculation of *shift(j)* and therefore *next (j).*

7. EXPERIMENTAL RESULT

To measure performance, we count the number of passes over the same input element while tested against a pattern element when we ran OPS* and compared the result with the number of passes in naïve search algorithm. The speedups obtained range from the modest simple search pattern without any recurring sub-pattern, to speedups of more than two orders of magnitude obtained on the complex patterns found in actual applications. For instance, Example 3, looks for a consecutive interval of period of high fluctuation (more than 1% up and down) followed by a period of steady raise in the stock price in DJIA (Dow Jones Industrial Average) index for 1975-2000.

Example 3. Pattern of consecutive interval of period of high fluctuation (more than 1% up and down) followed by a period of steady raise.

```
SELECT X.NEXT.date, X.NEXT.price,
          T.previous.date, T.previous.price
FROM djia
          SEQUENCE BY date
          AS (*X,*(*Y, *Z), T)
WHERE X.price >=  X.previous.price
          AND Y.price < 0.99 * Y.previous.price
          AND Z.price > 1.01*Z.previous.price
          AND T.price <= 1.01 * T.previous.price
```

While running OPS*, we found 32 matches over 25 years of DJIA data. **Figure 3** shows on the patterns that occurred around December 2000. The speedups we obtained from running several queries were up to 100 times.

8. IMPROVING AND EXTENDING OPS*

Although our preliminary result is encouraging and shows more expressive power plus an impressive speedup up to 100 times, we are still working on new techniques to get even more speedup. Furthermore we are investigating on the requirements for extending OPS* to employ optimize pattern search over the stream data. Following, we briefly describe our current studies.

8.1 Pattern Dependent Search

As one possible technique, we are investigating on increasing the efficiency of optimization by compromising *accuracy* of search by employing statistical pattern learning in search. We can ignore a series of predicates comparison if with a high probability such comparison will fail or matches. Penalty of guessing a match incorrectly, introduces wrong answers that can be filtered out later. On the other hand, guessing a failure incorrectly, leads to loosing some of the potential answers. The learning rules are constructed as times series sequential association rules in the following format:

$$p_j \wedge q_k \Rightarrow r_l$$

Pattern Dependent Search finds a series of rules after observing enough data from data stream.

8.2 Buffer Size

Considering that most streaming applications have a limited buffer size for the transient data, in order to make OPS* applicable to streaming applications, we need to address the buffer size issue. Since OPS* minimizes repeated passes over the same data by precomputing *shift(j)* and *next(j)*, it is obvious that in every pass we can remove some of the input data from the buffer and read the same amount of new incoming input data to the buffer. Our goal at this stage is to find an optimal buffer size for useful pattern classes.

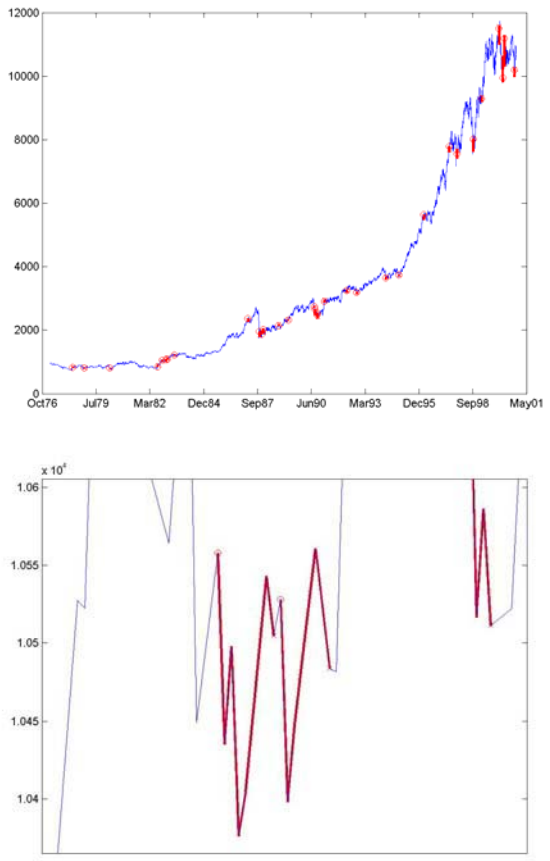


Figure 3. 32 consecutive bump shape fluctuations found in the DJIA data are shown in red. The bottom picture is zoomed for the area pointed by arrow in the top picture and shows one of the matches.

8.3 Data Compression

We are also interested in data compression techniques based on the pattern elements. For instance suppose the pattern we are looking for is *(X,*Y,*Z) and $p_2(t) = (t.price = 5)$. Now suppose in the input stream, there are six consecutive 5 matching $p_2(t)$. As $p_2(t)$ is a star element, we can keep only the index of the first of these consecutive fives instead of storing all six of them. Even though sampling and compression is addressed in stream data literature, there are new aspects such as information stored in the pattern itself, which needs to be addressed in this body of work.

8.4 Multi Dimensional Data Streams

Our next effort is extending OPS* algorithm to be able to search for complex pattern in multi- dimensions data streams such as video streams, trees and graphs.

9. CONCLUSION REMARKS

In this paper, we described OPS* algorithm which gives us a very expressive powerful tool to look for complex nested recurring patterns in data streams. By exploiting the inter-dependencies between the elements of a sequential pattern, we minimize repeated passes over the same data which impressively speedups the search process. We are currently investigating to make OPS* fully compatible with data stream characteristics.

10. ACKNOWLEDGMENTS

This research was supported in part by the USC Integrated Media Systems Center, a National Science Foundation Engineering Center, cooperative agreement No. EEC-9529152.

11. REFERENCES

1. Agrawal, R., Faloutsos C. and Swami, A. *Efficient similarity search in sequence databases*. in *The fourth International Conference on Foundataion of Data Organization and Algorithm*. 1993.
2. Agrawal, R., Lin, K. I., Sawheny, H. R., and Shim, K. *Fast similarity search in the presence of noise, scaling and translation in time series databases*. in *VLDB*. 1995.
3. Agrawal, R., Srikant, R. *Mining sequential pattern*. in *ICDE*. 1995. Taiwan.
4. Babcock, B., Babu, S., Datar, M., Motawani, R., and Widom, J. *Models and issues in data stream systems*. in *Proceedings of the Twenty first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. 2002.
5. Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N. and Zdonik, S. *Monitoring Streams: A New Class of Data Management Applications.* in *In proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02),*. 2002. Hong Kong, China.
6. Chandrasekaran, S., Cooper,O., Deshpande, A., Franklin, M., Hellerstein, J., Hong, W., Krishnamurthy, S., Madden, S., Raman V., Reiss, F., Shah, M. *TelegraphCQ: Continuous Dataflow Processing for an Uncertain World*. in *CIDR*. 2003.
7. Dalvi, N., Sanghai, S., Roy, P., Sudarshan, S., Pipelining. *Multi-Query Optimization*. in *PODS*. 2001.
8. Das, G., Lin, K., Mannila, H., Renganathan, G. and Smyth, P. *Rule discovery for time series*. in *KDD*. 1995. New York City, New York.
9. Gehrke, J.E., Korn, F., and Srivastava, D. *On Computing Correlated Aggregates Over Continual Data Streams.* in *ACM Sigmod International Conference on Management of Data*. 2001. Santa Barbara , CA.
10. Gupta, A., Sudarshan, S., Viswanathan, S. *Query Scheduling in Multi Query Optimization*. in *IDEAS*. 2001.
11. Informix Software, I., *Managing time-series data infinancial applications*. 1998.
12. Keogh, E., Smyth, P. *A Probabilistic Approach to fast pattern matching in time series databases*. in *KDD*. 1997.
13. Knuth, D.E., Morris, J. H., and Pratt, V. R., *Fast pattern matching in strings.* SIAM Journal of Computing, 1997. **6**(2): p. 323-350.
14. Mannila, H., Toivonen, H. and Verkamo, A. *Discovering generalized episodes using minimal occurance*. in *KDD*. 1996. Portland, Oregon.
15. Perng, C.-S.a.P., D. S. *Sql/lpp: A time series extension of sql based on limited patience patterns*. in *Database and Expert Systems Applications, 10th Int. Conf., DEXA*. 1999: Computer Science, Springer.
16. Ramakrishnan, R.e.a. *SRQL: sorted relational query language*. 1998. SSDBM.
17. Roy, P., Seshadri, A., Sudarshan, A., Bhobhe, S. *Efficient and Extensible Algorithms For Multi Query Optimization.* in *SIGMOD*. 2000.
18. Sadri, R., Zaniolo, C., Zarkesh, A., Adibi, J. *Optimization of pattern matching Queries on Database Sequences*. in *PODS, Twentieth ACM SIGACT-SIGMOD*. 2001. Santa Barbara,USA.
19. Sellis, T., *Multiple Query Optimization.".* ACM TODS, 1988. **13**(1): p. 23-52,.
20. Seshadri, P. *Predator: A resource for database research*. 1998: *SIGMOD Record*.
21. Seshadri, P., Livny, M. and Ramakrishnan, R. *Sequence query processing*. in *In Proc. of ACM SIGMOD Conference on Management of Data*. 1994.
22. Seshadri, P., Livny, M. and Ramakrishnan, R. *SEQ: A model for sequence databases*. in *ICDE*. 1995.
23. Smyth, P., *Clustering sequences using Hidden Markov Models.* Advances in Neural Information Processing, 1997.
24. The STREAM Group, *STREAM: The Stanford Stream Data Manager*. *IEEE Data Engineering Bulletin*, March 2003. **26**(1).
25. Tsong-Li Wang, J., Chim, G., Marr, T.G., Shapiro, B. A., Shasha, D., and Zhang, K. *Combinatorial pattern discovery for scientific data: Some preliminary results*. in *SIGMOD*. 1994.
26. Widom, et al., *Continuous Queries over Data Streams.* SIGMOD Record, 2001.
27. Yang, J., and Widom, J. *Temporal View Self-Maintenance*. in *ACM TODS*. 2000.
28. Yang, J., AND Widom, J. *Incremental Computation and Maintenance of Temporal Aggregates*. in *ICDE*. 2001.

A note on predicting corporate default from the bond market

Abhinanda Sarkar
GE Global Research

GE John F. Welch Technology Center
EPIP Phase 2, Whitefield Road
Bangalore 560066, India

Abhinanda.Sarkar@geind.ge.com
91-80-2503-2775

Debasis Bal
GE Global Research

GE John F. Welch Technology Center
EPIP Phase 2, Whitefield Road
Bangalore 560066, India

Debasis.Bal@geind.ge.com
91-80-2503-2427

Abstract

We present a brief description of the methods behind an early warning system for predicting corporate default. The data we use comes from the US bond market and is in the form of daily data streams of corporate bond prices, yields, and maturities as well as corresponding US treasury bond metrics. Default probabilities are estimated for companies based on the spread of the bond portfolio over risk-free government bonds. In order to do this matching, a simple and efficient term structure model is built. In addition to spreads, volatility is incorporated making a novel use of a return-to-variability (Sharpe) ratio of bond prices. Together the two metrics define a risk space through which a temporal trace of the risk profile of companies can be obtained. An operational real-time system that collects bond data from the Web has been built based on the method and is described. We illustrate our themes through four companies whose bond portfolios showed different characteristics during 2002-2003.

I. Introduction

In the period from 2001 to 2003, due to the prevailing business climate at the time, companies were more than typically susceptible to defaulting on their financial obligations. In order to maintain healthy and well-controlled investment portfolios, being able to adequately predict corporate default became of paramount importance for investors and lending agencies.

This note outlines the methods and system for such a default prediction system based on a temporal analysis of the bond market. Of course there has been a fair amount of work and discussion on the efficacy of data mining methods on financial markets, such as the stock market [1], but the bond market poses distinct challenges, not the least of which is paucity of both real-time as well as historical data.

Organizations such as KMV (now Moody's KMV) provide methods and systems for early warning of corporate default based on the equity (stock) market [4]. The data is principally in the form of stock prices of publicly traded companies. Less work has been done on the bond market [7], mainly because (a) fewer companies issue bonds and (b) the debt market is perceived as being less predictive when it comes to default. The reason for the latter is the truism that, on default, bondholders are repaid before stockholders and thus bond market will react less to indications of impending distress. Our methods and system are not designed to replace stock market based technology, but rather to support and append them. We restrict ourselves to the US market.

This note is also about constructing a temporal database that can store and enable post-processing of real-time bond market data, though it must be admitted that such databases are not "large" by current standards in data mining.

In sections II through V, a method is described by which we estimate the probability of default for a company based on daily bond yields. The fundamental premise is that corporate bonds have higher open-market yields than their government counterparts. This observed difference can be extracted from

observed data (after some smoothing) and converted to a default probability estimate.

Sections VI and VII describe an additional measure based on bond price volatilities. Together the two measures define a so-called risk space that tracks the performance of companies.

Section VIII provided only the outlines of an operational risk management system that we have built implementing these ideas.

The financial and Web technologies reported in this work are not, by and large, novel. The exceptions are as follows, with details in the subsequent sections.

1. The use of a fast approximation to fit real-time term structures for treasury bonds and the use of the fit to match bond maturities.
2. The definition a return-to-variability (or Sharpe) ratio for bond prices with specific use as a default indicator.
3. The construction of a Web agent for use as a bond market corporate risk dashboard.

II. Probability of default

Corporate bonds have higher yields than bonds issued by governments, the US government in our case. This primarily to reflect the possibility of default, i.e. the bond may not actually give the yield if the company goes into financial distress. Thus the difference between the yields of corporate and treasury bonds provides a metric for estimating default probabilities. See [6,10] for more details.

A present value computation provides a rationale for our choice of estimate of default probability. Let y be the yield of the corporate bond of maturity t and let $r(t)$ be the yield of a treasury bond of the same maturity. If the face value is 100, the present value of the corporate bond is $100e^{-y}$ and the corresponding treasury present value is $100e^{-r(t)}$. Denote the probability of default by p. We make the assumption that there is complete loss given default. In other words, if the company defaults on its bonds, bondholders do not get any returns from the yield-generating part of the debt product. With that assumption, the expected present value from the corporate bond is $p\times 0+(1-p)\times 100e^{-y}$. The bond's yield will be such that this is the same as that from the treasury. Here we assume that the higher yield from the corporate bond is entirely due to the risk of default. (This is not really the case [10], but a more refined estimate requires financial data of altogether different kinds.) Thus we obtain the identity $(1-p)e^{-y}=e^{-r(t)}$ and we deduce that $p=1-e^{-(y-r(t))}$. The risk premium captured by $y-r(t)$ is called the spread and we denote it by s. The probability of default, using the mentioned assumptions is thus estimated by $p=1-e^{-s}$.

An example modified from [7] illustrates this in slightly different terms.

Suppose that a 5-year treasury bond with a face value of 100 yields 5% per annum and a similar 5-year corporate bond yields 5.5% per annum. The value of the treasury bond is

$$100\times e^{-0.05\times 5}=77.88$$

and the value of the corporate bond is .

$$100\times e^{-0.055\times 5}=75.96$$

The present value of the cost of default is, therefore, 77.88 - 75.96 = 1.92. Denote the probability of default during the 5-year life of the bond by *p*. If we make the simplifying assumption that there are no recoveries in the event of a default, the impact of a default is to create a loss of 100 at the end of 5 years. The expected loss from default is therefore $100\times p$ and the present value of the expected loss is $100\times p\times e^{-0.05\times 5}$. It follow that this expected loss equals 1.92 so that p = 0.0247 or 2.47%.

The driver for default probability is thus the risk premium captured by the spread, i.e. the difference between the corporate bond yield and the risk-free treasury rate.

For a typical low-risk company, here is what the spread looks like. The lower curve of matching treasury yields is smoothed using a method described in the next section.

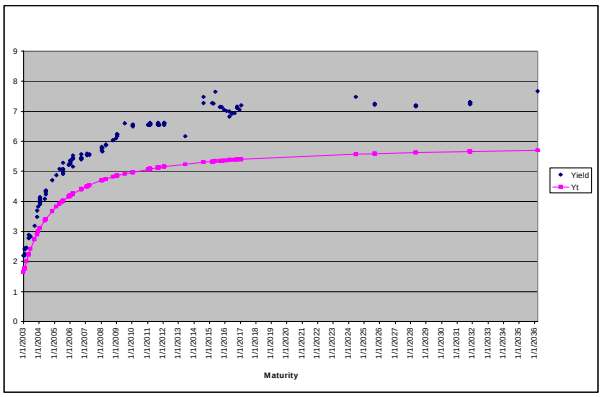


Figure 1

Departures from this general shape, as we shall subsequently illustrate, indicate financial distress.

III. Modeling the treasury term structure

Our method requires the matching of corporate bonds with treasury bonds of the same maturity. As such matching treasury bonds may not be trading at that time, a smoothing model is needed to determine interest rates for arbitrary maturity dates or terms.

There are a great many models for the term structure of interest rates. For our purposes, we select a particularly simple one of the mean reverting type, derived from [3]. Let $r(t)$ denote the treasury yield at maturity t. Our model postulates

$$\frac{d}{dt}r(t) = \alpha[r_{\max} - r(t)] + \eta(t)$$

where $\alpha > 0$ is a rate of convergence to the long-term rate $r_{\max}$ and η is a noise term. We fit this model using a discrete approximation described below and then take as the matching treasury rate, the fitted yield that corresponds to the maturity of the corporate bond under investigation.

For the purposes of this note, we ignore the nature of the noise and describe an approximate algorithm to estimate parameters. The mean reverting differential equation has as a solution $r = r_{\max}(1 - e^{-\alpha t})$. A possibility is to directly fit a nonlinear regression of this form to the yield data. An alternative is to consider the approximation $r \approx r_{\max}\frac{\alpha t}{1+\alpha t}$, which is a mean reverting model in its own right. Recasting the model as $\frac{1}{r} \approx \frac{1}{r_{\max}} + \frac{1}{\alpha r_{\max}} \times \frac{1}{t}$ we see that a linear regression of r^{-1} on t^{-1} suffices to fit the model. This is the fitting strategy we use to estimate a term structure for the treasury. (Note: The use of reciprocal regressions to estimate relationships that have a horizontal asymptote is familiar in chemical kinetics via the so-called Michaelis-Menten model. See, for example, [8].)

Here is an example of such as fit with the actual interest rates (yields) indicated as points and the fit as a line.

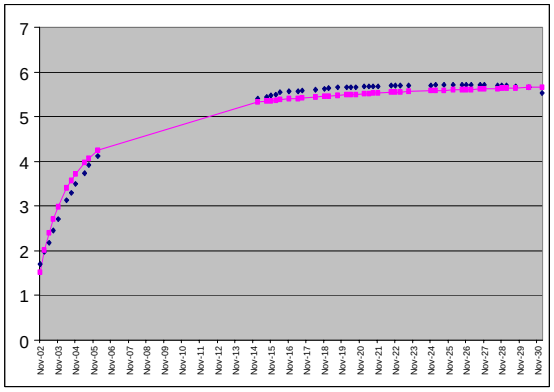


Figure 2

IV. Examples

We provide four examples of companies demonstrating four types of temporal behavior of the yield curve over the course of a year. All bonds for the company being traded on a particular day in the month indicated are shown yield versus time to maturity in years. The estimated default probabilities as computed by the system are plotted. The variability in the estimated default probabilities has business connotations that we report on elsewhere.

Note that the average spread of this over the treasury rate determines the estimated default probability. The treasury rates for the corresponding days are also plotted in a separate graph.

Figure 3 : Citigroup: a company that stayed at low risk throughout the period.

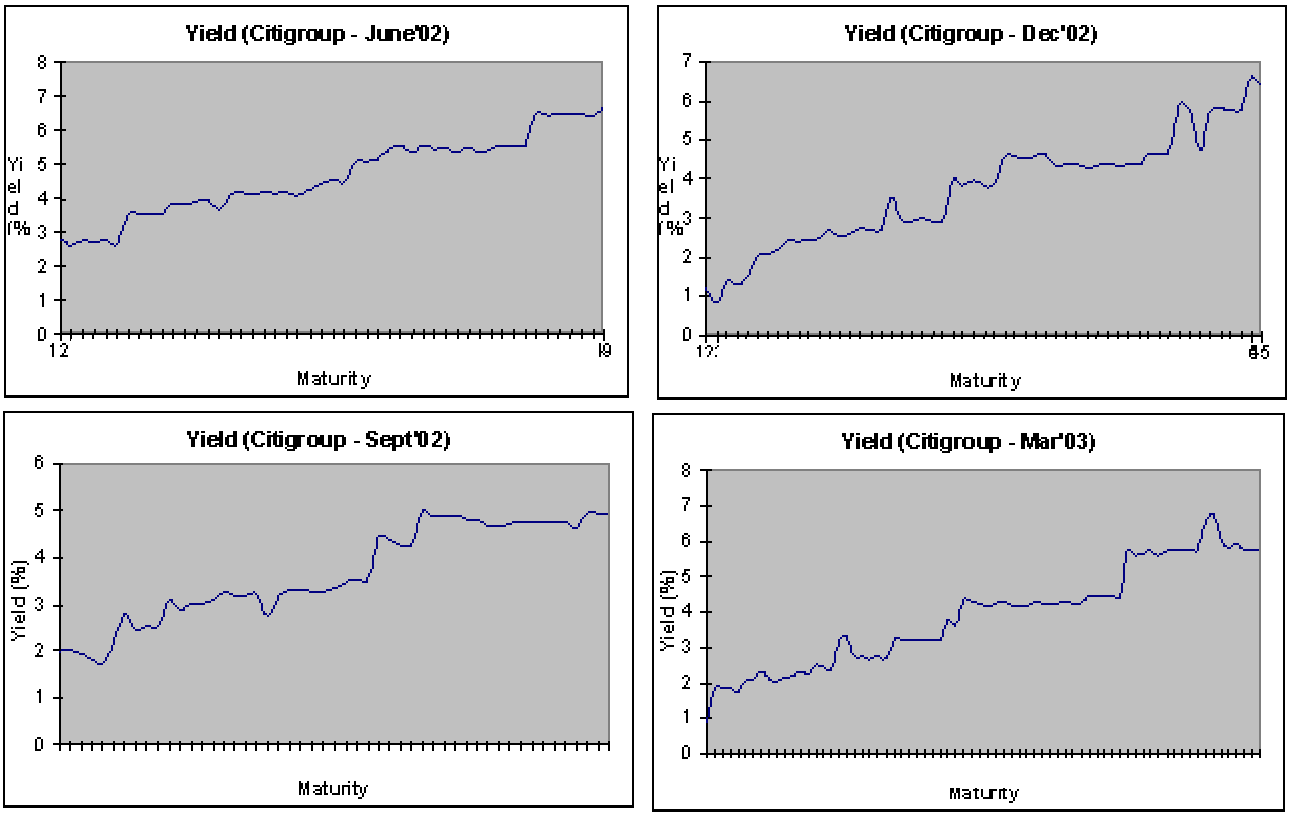
Figure 4 : Tyco: a company that moved from high risk to low risk over the period.

Figure 5 : Worldcom: a company that remained at high risk during the period (and for which we correctly predicted default).

Figure 6 : Northwest: a company that went from relatively low risk to high risk over the period.

Figure 7 : Treasury yield curves at corresponding times and a temporal plot of treasury yields averaged over maturities.

Figure 3


Yield (Citigroup - June'02)
Yield (%)
Maturity
Yield (Citigroup - Dec'02)
Maturity
Yield (Citigroup - Sept'02)
Yield (%)
Maturity
Yield (Citigroup - Mar'03)
Yield (%)
Maturity

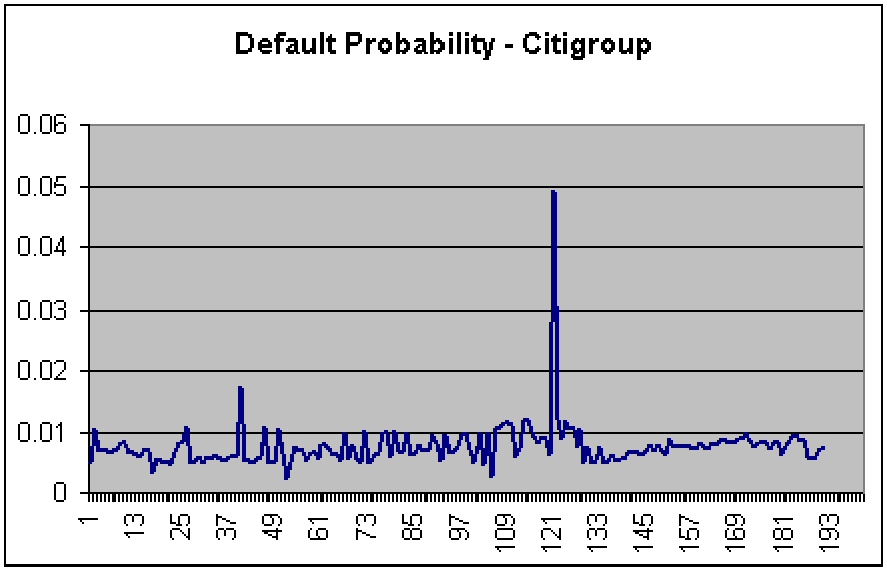
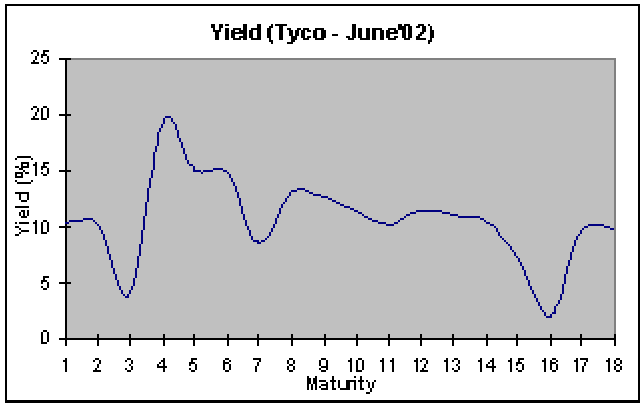
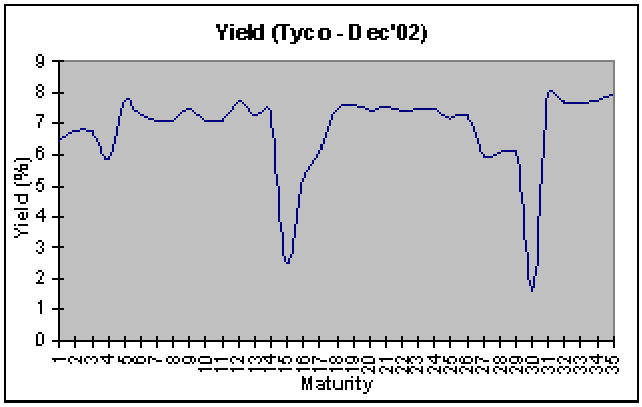
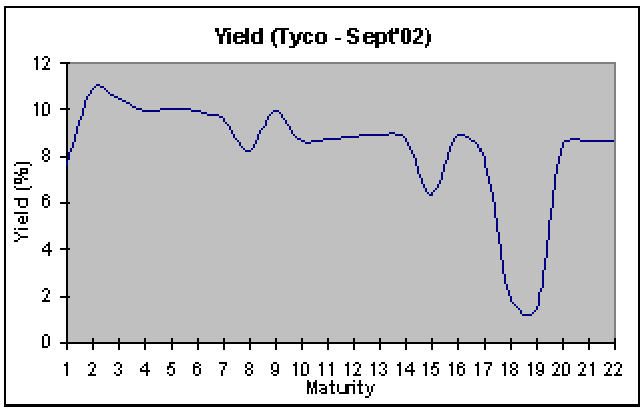
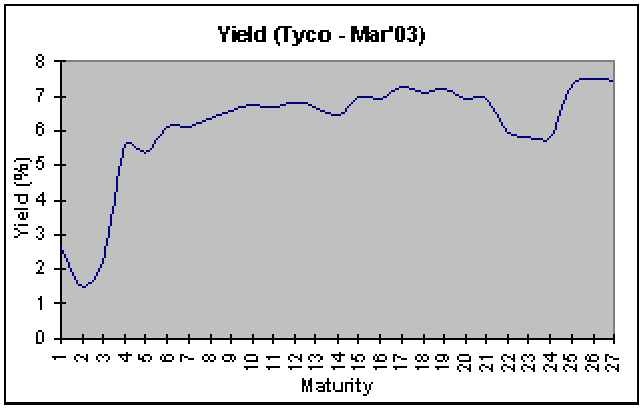

Default Probability - Citigroup
0.06
0.05
0.04
0.03
0.02
0.01
0
1
13
25
37
49
61
73
85
97
109
121
133
145
157
169
181
193

Figure 4


Yield (Tyco - June'02)
Yield (%)
Maturity


Yield (Tyco - Dec'02)
Yield (%)
Maturity


Yield (Tyco - Sept'02)
Yield (%)
Maturity


Yield (Tyco - Mar'03)
Yield (%)
Maturity

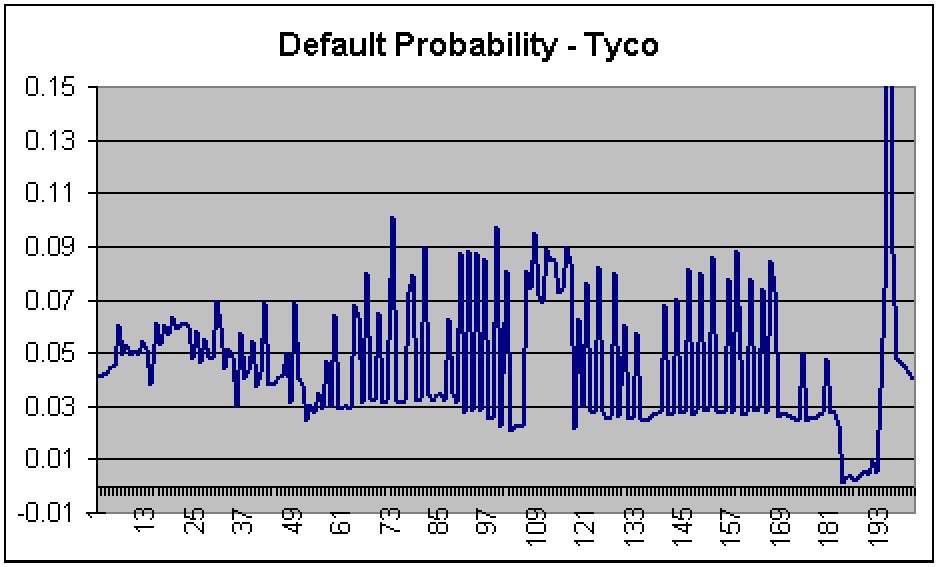
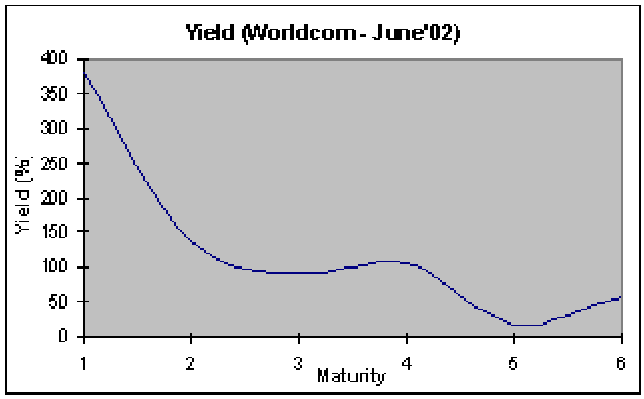
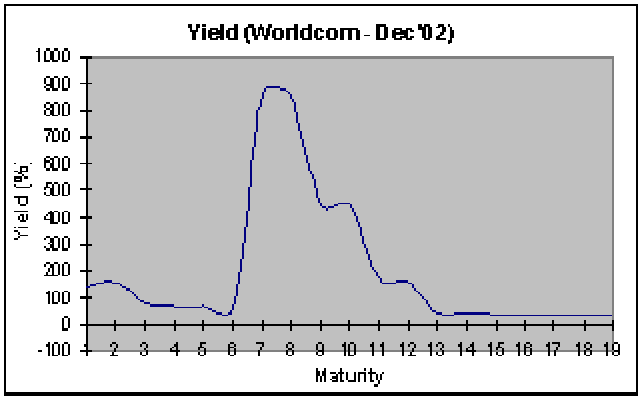
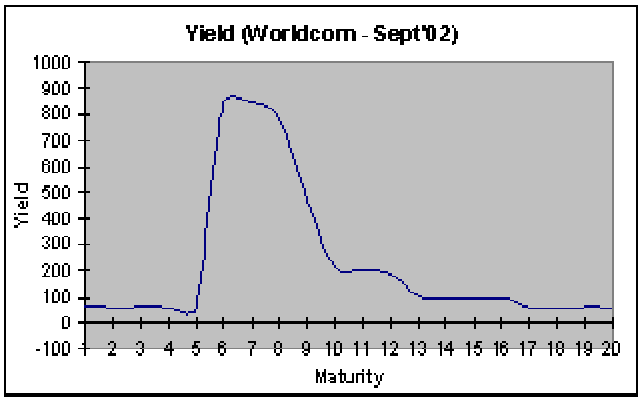
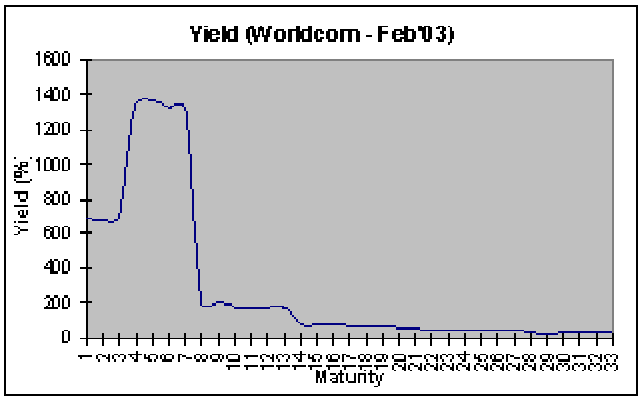

Default Probability - Tyco

Figure 5


Yield (Worldcom - June'02)
Yield (%)
Maturity


Yield (Worldcom - Dec'02)
Yield (%)
Maturity


Yield (Worldcom - Sept'02)
Yield
Maturity


Yield (Worldcom - Feb'03)
Yield (%)
Maturity

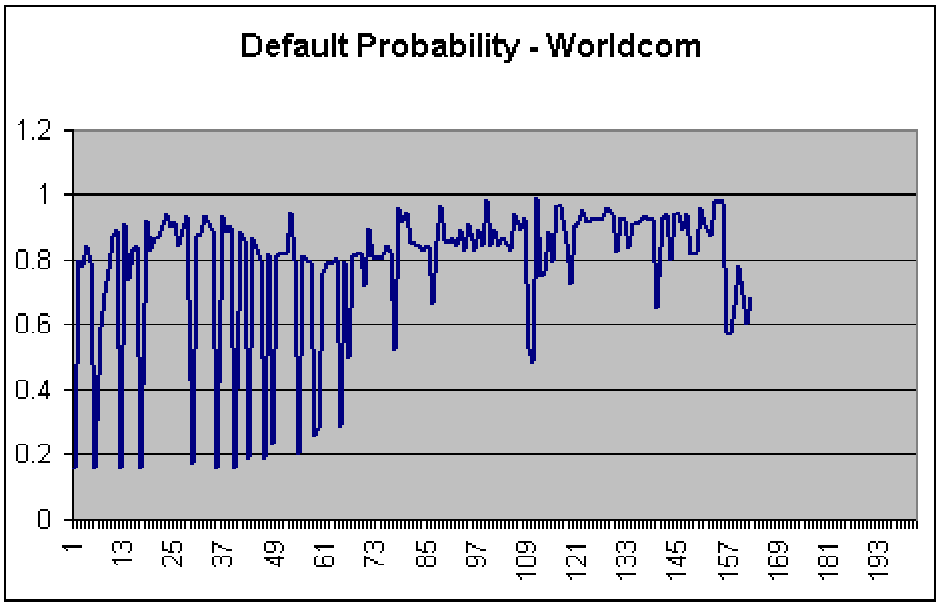
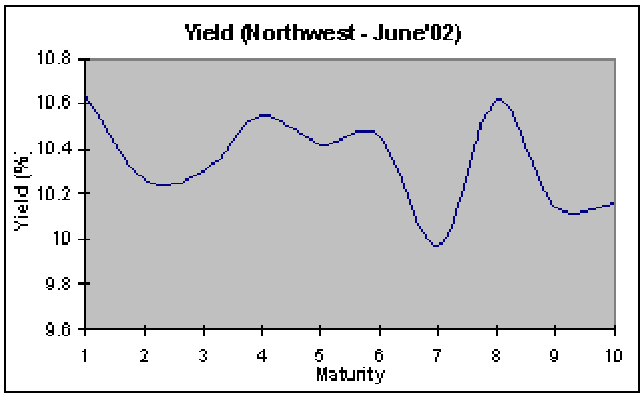
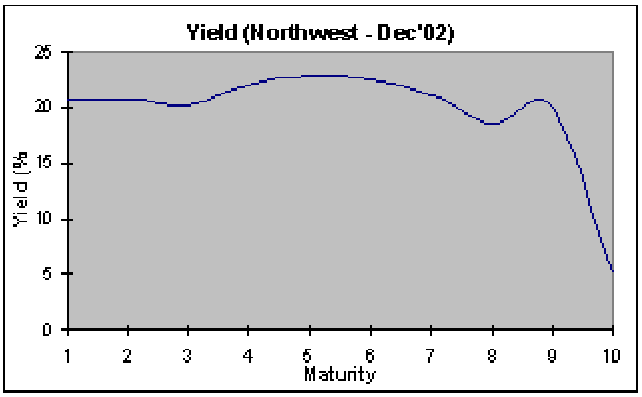
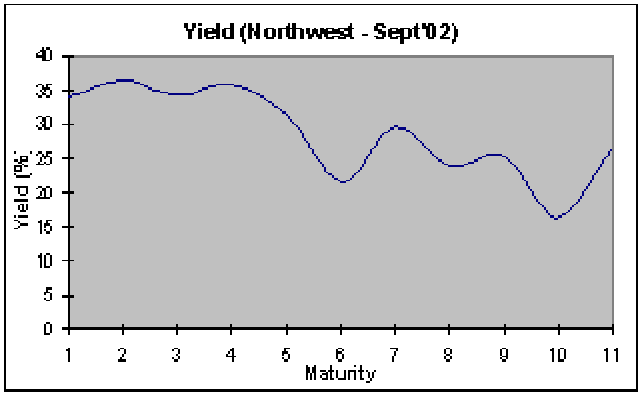
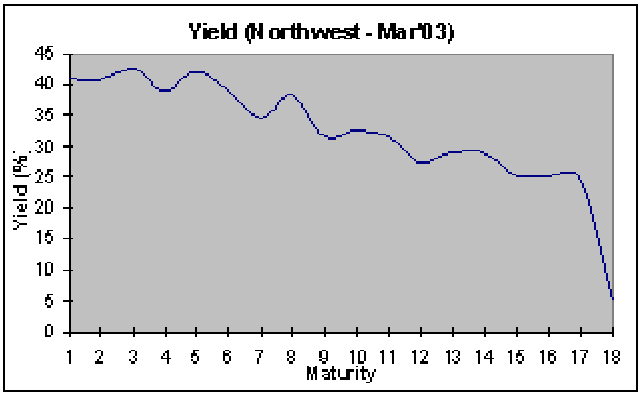

Default Probability - Worldcom

Figure 6


Yield (Northwest - June'02)
Yield (%)
Maturity


Yield (Northwest - Dec'02)
Yield (%
Maturity


Yield (Northwest - Sept'02)
Yield (%)
Maturity


Yield (Northwest - Mar'03)
Yield (%)
Maturity

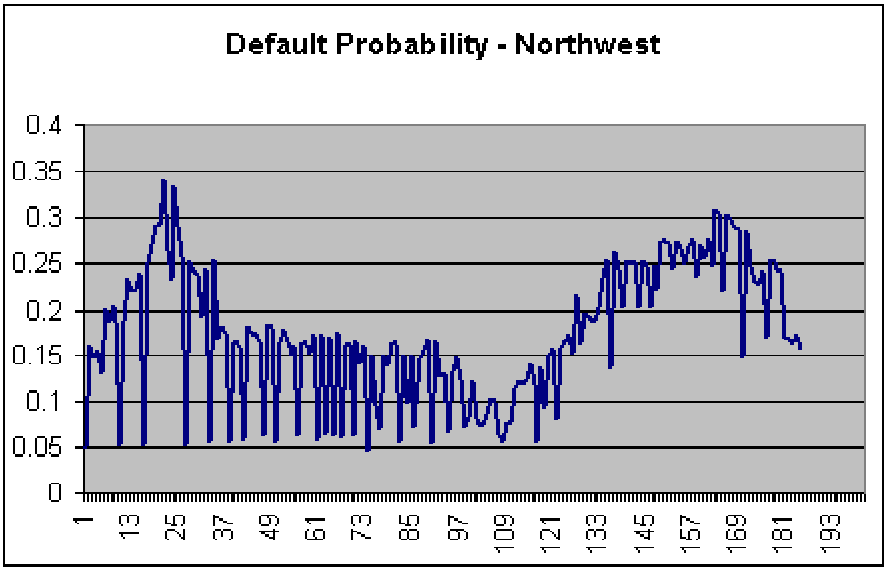
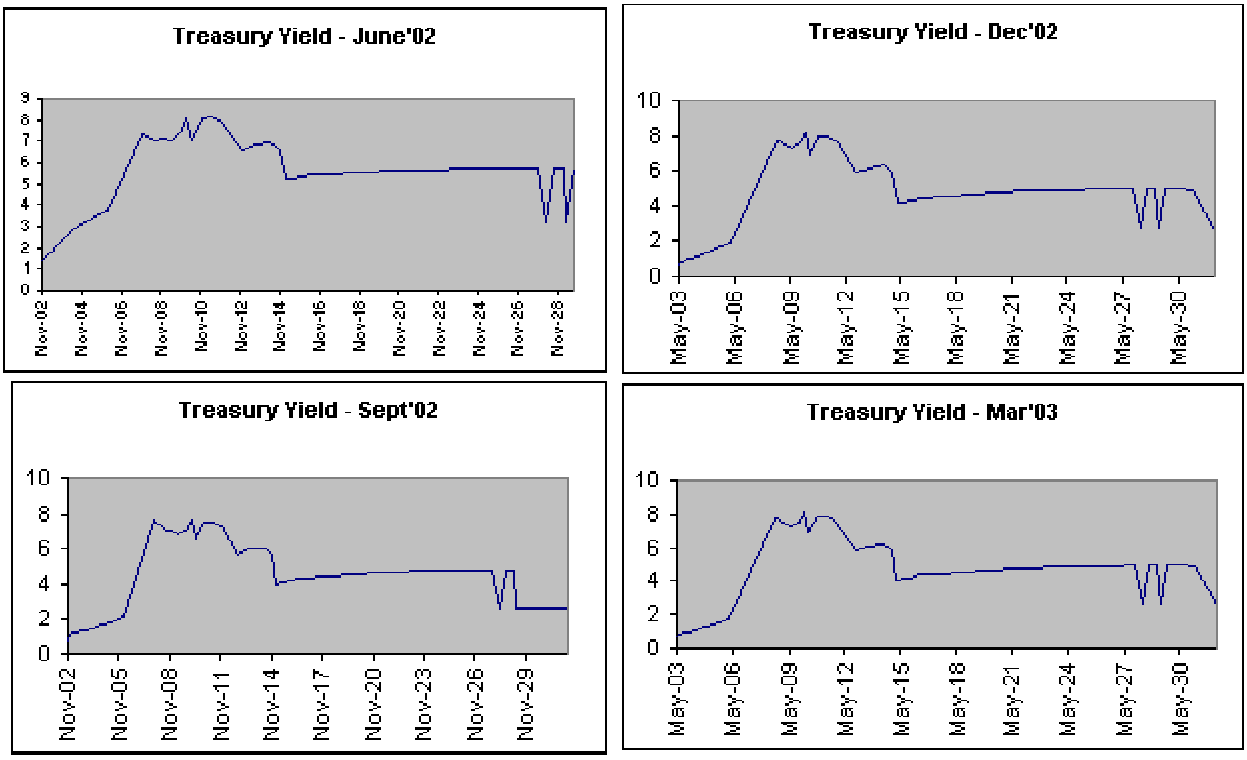
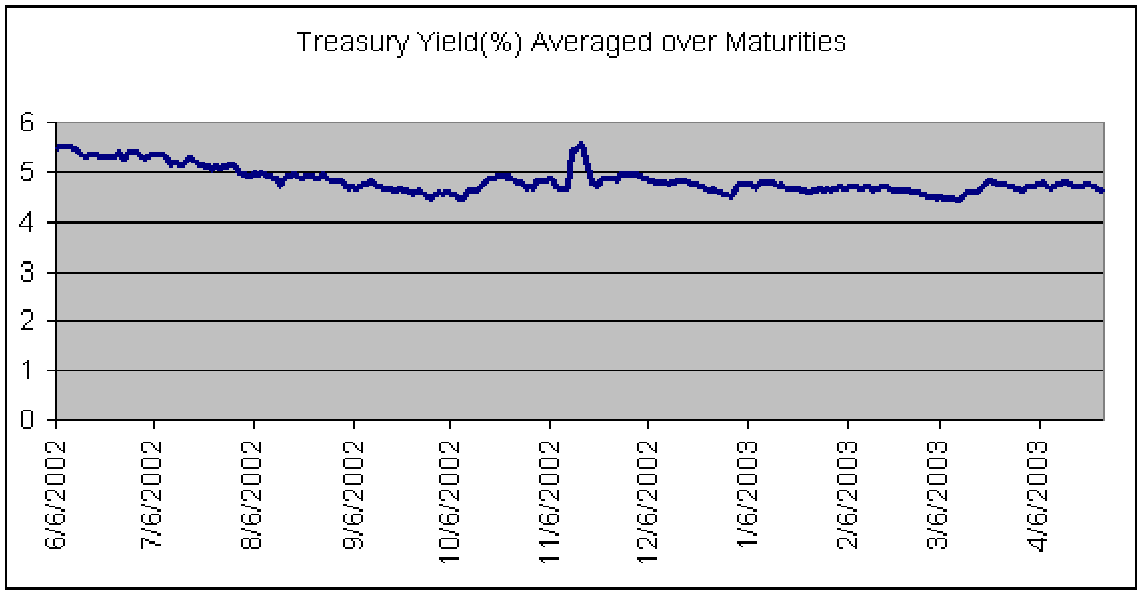

Default Probability - Northwest

Figure 7


Treasury Yield - June'02
Treasury Yield - Dec'02
Treasury Yield - Sept'02
Treasury Yield - Mar'03


Treasury Yield(%) Averaged over Maturities
6
5
4
3
2
1
0
6/6/2002
7/6/2002
8/6/2002
9/6/2002
10/6/2002
11/6/2002
12/6/2002
1/6/2003
2/6/2003
3/6/2003
4/6/2003

V. Bond portfolios

On a given day, a particular company may have several bonds being traded of varying maturities, yields, and prices. We have tried several maturity-indexed schemes to combine the default signals in the multiple bonds. The simplest one gave the best predictive power, namely consider the unweighted average of the spreads of the various bonds and convert as above to a default probability estimate of the company on the given day. Let $y_1, y_2, \ldots y_B$ be the yields of *B* bonds of a company with maturities $t_1, t_2, \ldots t_B$ respectively. Let $r(t_1), r(t_2), \ldots r(t_B)$ be the yields of treasury bonds with the same maturities. It is unlikely that such bonds will trade on a particular day, so we use the fitted term structure model to find the matching yield. The default probability of the company is estimated as $p = 1 - e^{\bar{s}}$ where

$$\bar{s} = \frac{1}{B}\sum_{i=1}^{B} \max[y_i - r(t_i), 0]$$

Note that while we are calling this estimate a default probability, it carries with it a multitude of simplifying assumptions that make its use an actual likelihood of default improper. However, as we are comparing companies and free to choose empirical thresholds, such a crude estimate suffices for our purposes. Machine learning algorithms reported elsewhere determine the empirical thresholds for our operational system.

VI. Volatility and a risk space

In addition to default probability, another dimension to risk is provided by volatility or variation in bond yields and prices.

The use of volatility to measure risk is a fundamental paradigm in finance. A security with a more variable price or value is judged to be at higher investment risk among those with similar returns. We enhance our early warning system based on the default probability measure by incorporating the volatility of bond prices. The measure we choose is a return-to-variability or Sharpe ratio of the prices of the bonds in a company's portfolio.

The Sharpe ratio is a measure of risk that is in common use in investment management [10]. The basic premise has long been a part of investment folklore. Nobel laureate William Sharpe espoused its particular use as a measure in the mathematics of asset pricing. In its most elementary form, the Sharpe ratio of a financial asset with a variable return is defined by

$$\frac{\mu - \mu_0}{\sigma}$$

where μ is the mean return of the asset and σ is the variance of the returns. μ_0 is the risk-free rate of return in the market, i.e. the return of a product that has no uncertainty (zero variance). The Sharpe ratio is a risk-calibrated measure of the quality of returns of the asset and is particularly useful as a guide to performance-driven investors.

A Sharpe ratio that we are proposing is the ratio of mean and standard deviation of bond prices. A high-risk company will have low prices and variable prices, both of which contribute to a low Sharpe ratio. If $P_1, P_2, \ldots P_B$ are the prices in the portfolio, define

$$SR = \frac{\overline{P}}{\sqrt{\frac{1}{B-1}\sum_{i=1}^{B}(P_i - \overline{P})^2}}$$

where

$$\overline{P} = \frac{1}{B}\sum_{i=1}^{B} P_i$$

Low values of *SR* are indicative of high default potential. Of course, this measure can only be computed if there are multiple bonds in the market for the company. An alternative definition that we have found to have a little more predictive power is to consider price normalized by the treasury price. In other words, instead of P_i, define the *SR* using

$$\tilde{P}_i = \frac{P_i}{P(t_i)}$$

where $P(t_i)$ is the price of the treasury bond of the same maturity. This requires using the fitted term structure if there are no matching treasury bonds.

Thus for a company, we consider a pair of default risk measures (p, SR). On a specific day, we plot this pair of values for each company to obtain the risk space for the day. The companies in the high *p* low *SR* corner are those we should keep a close eye on for signs of default. In our experience, the *p* signal is stronger than the *SR* signal, bit there are cases where the latter has been surprisingly effective.

The traces in the two-dimensional risk space for our example companies is presented showcasing its use as an evolutionary predictive tool. The "danger zone" is also indicated.

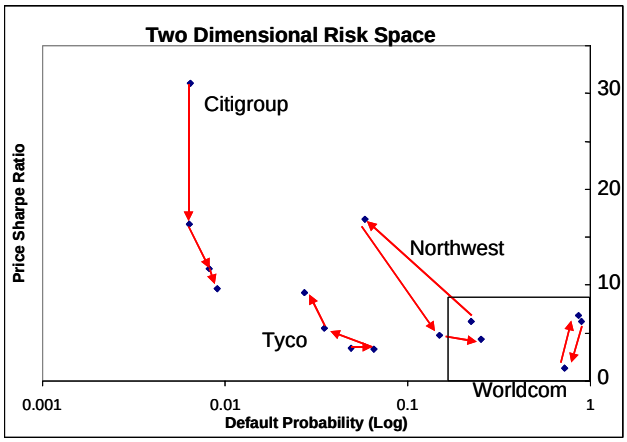


Figure 8

VII. Bond prices and yields

The price of a bond is, in fact, directly related to the yield as the following shows [3].

For our purposes, we take the face value of a bond as 100. If the bond has maturity time T and annual payout (coupon rate) c, then the price of the bond P and the yield y are related by equating discounted cash flows till maturity with price, i.e.

.

$$P = \frac{c}{(1+y)} + \frac{c}{(1+y)^2} + \cdots + \frac{c}{(1+y)^{T-1}} + \frac{c+100}{(1+y)^T}$$

In the simplifying case of a zero-coupon bond, we have the more familiar

$$P = 100(1+y)^{-T}$$

Yield and price are thus inversely related, and a risky bond has high risk premium (captured by high yield) and a corresponding low price.

VIII. System

We have built a system (see Figure 9) that implements the method of early prediction of corporate defaults using the bond market information as described in this note. It has three main components.

1. A web agent that collects daily corporate bond and treasury bond data from the Internet. We use Internet as our source of daily traded bond data. In another implementation a data repository may replace the Internet. On a specific day we look at the bonds for the company and record (a) yield, (b) price, and (c) maturity.

2. A data-parsing engine that cleans and formats the downloaded data into a format suitable for analysis and appends it to a growing database.

3. An analytics engine that computes the estimated default probability and the volatility as measured by the Sharpe ratio and appends the same to the database.

For each company these daily records are then fed into a temporal map that traces over the days the presence of the company on the two-dimensional risk space. Companies drifting into the danger zone are flagged for further investigation. In addition, other customized forms of daily reports are generated depicting the default behavior of the companies.

In effect, this is a use of temporal data mining for development of an early warning system for corporate default. Of course, while such real-time Webcrawlers are not new as financial decision support systems [2,5,9], the urgency of the corporate default early warning problem makes the implementation challenge currently of critical importance.

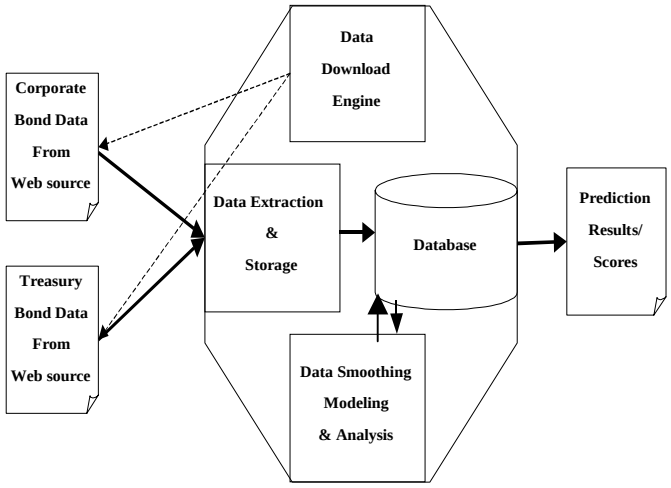


Figure 9

IX. Acknowledgements

We thank our colleagues in GE Global Research, particularly Roger Hoerl (who provided the inspiration for the project) and Gopi Subramanian (who helped greatly with the programming). We are also grateful for helpful comments from test users of the system in GE Global Risk Management and workshop participants at the Tata Institute of Fundamental Research.

X. References

[1] "Can data mining predict the S&P 500?" http://www.kdnuggets.com/polls/2002/predicting_S&P500.htm

[2] Atzeni, P., Mecca, G., and Merialdo, P. (2002) "Managing Web-based data – database models and transformations", IEEE Internet Computing, vol 6, issue 4.

[3] Benninga, S. (1997) "Financial Modeling". MIT Press.

[4] Crosbie, P and Bohn J. (2002) "Modeling Default Risk". White paper. KMV.

[5] Decker K., Sycara, K., and Williamson, M. (1997). "Middle-Agents for the Internet", 15th IJCAI.

[6] Hull, J. (1997) "Options, Futures, and other Derivatives". Third Edition. Prentice-Hall.

[7] Hull, J. and White, A. (2000) "Valuing Credit Default Swaps I: No Counterparty Default Risk". Working paper. University of Toronto.

[8] Moran, L., Scrimgeour, K, Horton, H., Ochs, R., and Rawn, J. (1994) "Biochemistry". Second Edition. Prentice-Hall.

[9] Sarkar, A. and Ananthanarayanan, R. (2001) "Learning from E-Commerce: Incomplete Data and Decision Support", Bulletin of the ISI, 53rd Session, Seoul.

[10] Sharpe, W, Alexander, G, and Bailey, J (1998), "Investments". Sixth Edition. Prentice-Hall.