Proceedings of the 2007 IEEE Symposium on
Computational Intelligence and Data Mining (CIDM 2007)

Discovery of Temporal Dependencies between Frequent Patterns in Multivariate Time Series

Giridhar Tatavarty, Raj Bhatnagar, Barrington Young
CS Department, University of Cincinnati, Cincinnati, OH 45221, USA
Email: tatavag@ececs.uc.edu, raj.bhatnagar@uc.edu, younbn@ececs.uc.edu

Abstract— **We consider the problem of mining multivariate time series data for discovering (i) frequently occurring substring patterns in a dimension, (ii) temporal associations among these substring patterns within or across different dimensions, and (iii) large intervals that sustain a particular mode of operation. These represent patterns at three different levels of abstraction for a dataset having very fine granularity. Discovery of such temporal associations in a multivariate setting provides useful insights which results in a prediction and diagnostic capability for the domain. In this paper we present a methodology for efficiently discovering all frequent patterns in each dimension of the data using Suffix Trees; then clustering these substring patterns to construct equivalence classes of similar (approximately matching) patterns; and then searching for temporal dependencies among these equivalence classes using an efficient search algorithm. Modes of operation are then inferred as summarization of these temporal dependencies. Our method is generalizable, scalable, and can be adapted to provide robustness against noise, shifting, and scaling factors.**

I. INTRODUCTION

Multi-attribute time series data occurs in various domains such as finance, science and engineering applications, weather and network traffic monitoring. These datasets may contain repeated occurrences of some patterns with minor variability among different occurrences. Discovery of such repeating patterns provides insights into the domain's underlying processes. Important tasks in mining time series data address the issues of similarity search among different substring and subsequence patterns, discovery of frequently occurring patterns and the task of finding temporal associations among groups of frequently occurring patterns. Consider the following simple example. Weather monitoring data for a year may contain a intermittent but repeating pattern of temperature going up for three continuous days. It may also contain another repeating pattern of temperature falling for three continuous days. There may be other repeating patterns for buildup in pressure for a few days in a row or some particular variation pattern for humidity for a few days in a row. Our first objective is to discover these repeating patterns in each dimension (such as pressure, temperature, humidity). The second major objective is to discover temporal dependencies among the patterns occurring either within the same dimension, or across dimensions. Patterns are more meaningful, when they are associated with each other.For example,it is more informative to know: *"If temperature constantly rises for 4 days overlapped by a period of constant pressure for 5 days then after 3 days rainfall occurs and temperature, pressure drop".* than just knowing that: *temperature, pressure and rainfall follow particular patterns*. The first helps to establish correlations between patterns of underlying phenomena and enables us to develop predictive and diagnostic tools for the domain. As final summarization, we may want to identify say, the two 45-day long periods in a year during which this temporal association is observed. In this paper we present our methodology and demonstrate achieving all of these objectives.

II. RELATED WORK

Much work has been done in finding similar patterns in time series data, indexing the time series and mining them for repeating trends or patterns. Most such methods apply techniques [16] such as the Fourier and Wavelet Transforms, Dynamic Time Warping, Piecewise Approximation, Shape and Structural Grammars [14]. Various similarity measures [4] have been proposed to compare patterns in time series from different perspectives.

The problem of finding recurring substring patterns [7], [3], [8] has been solved using a number of approaches for periodic, non periodic and partially periodic patterns. The work described in [3] efficiently uses suffix trees to mine frequent patterns. The number of patterns invariably becomes very large. The approach in [7] formalizes a repeating pattern as a motif and presents EMMA algorithm to find k-motifs. Later a probabilistic algorithm was presented in [17] for the same. The work in [1] extends temporal association rule mining to interval based events and gives an algorithm for finding temporal dependencies. We consider more general types of temporal rules which include a subset of Allen's [5] all possible temporal relationships among events. [8] addresses a similar problem for multivariate time series by considering the qualitative characteristics. Our work is significantly different from the work presented in [8] as we identify temporal dependencies between similar patterns/shapes in time series while the work presented in [8] discovers only temporal dependencies between qualitative features of time series such as increasing or decreasing.

Contribution: Our primary contribution is an efficient methodology to discover temporal associations among patterns in a multi-dimensional time series dataset. It can tolerate significant noise levels in data, supports patterns of different lengths within one equivalence class, finds temporal association rules among the equivalence classes of string patterns

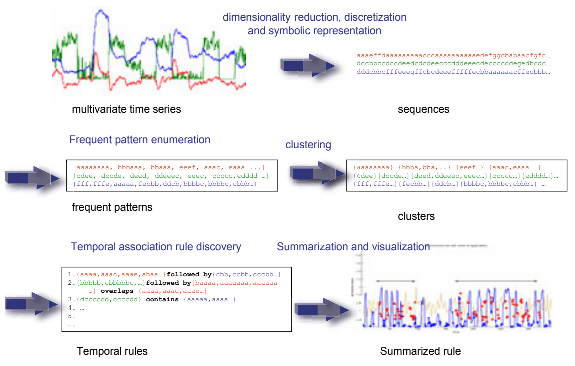
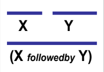
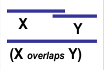
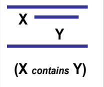

dimensionality reduction, discretization and symbolic representation
multivariate time series
sequences
Frequent pattern enumeration
clustering
frequent patterns
clusters
Temporal association rule discovery
Summarization and visualization
Temporal rules
Summarized rule

Fig. 1. Overview of temporal association rule discovery process

relation	function	pictorial example	end point constraints	resultant position pair
followedby	*followedby(X,Y)*		Xe < Ys Ys – Xe = WIN	(Xs,Ye)

overlaps	*Overlaps(X,Y)*		Xe = Ys
			Xs = Ys
			Xe = Ye
				(Xs, Ye)

contains	*Contains(X,Y)*		Xs < Ys Xe > Ye	(Xs, Xe)

X (Xs,Xe), Y (Ys,Ye) are two position pairs such that Xs = Ys

Fig. 2. Constraints for Temporal Relation between two position pairs X,Y

and summarizes. Our methodology goes beyond the ideas presented in [2] and [7] in different ways: (i)it efficiently identifies patterns of various lengths in a time series in the presence of noise (ii) it can find temporal associations between patterns of different lengths without the requirement of knowing the window-size of the pattern, (iii) it explores multiple temporal relations among patterns, and (iv) the temporal dependencies are summarized to identify stretches where the dependencies hold. figure 1 presents an overview of our methodology.

Paper Organization: The rest of this paper is organized as follows. In Section III we define the problem. In Section IV we discuss the process of normalization, dimensionality reduction, and quantization of the time series data to convert it to symbol strings. Section V describes our space efficient suffix-tree based search for frequent patterns in each dimension of the time series data and pruning techniques. Section VI presents clustering of frequently occurring substrings into a much smaller number of equivalent classes. Section VII presents the algorithm for discovering temporal dependencies between the equivalence classes of substrings. Section VIII presents the higher level summarization of temporal dependencies discovered in section VII. Section IX discusses the experimental results. Finally, we do a robustness analysis in Section IX for our methodology, and conclude in Section X.

III. Problem Definition

In this section we introduce the basic notions required to define and formulate the problem.

Definition 1: *A time series X is a finite sequence of real values* $(x_1, x_2, x_3 ... x_n)$. *A multi-dimensional time series in an m-dimensional space, with n observations in each dimensions, is represented by m sequences:*

$X_0 = (x_{00}, x_{01}, x_{02} ... x_{0n-1})$

$X_1 = (x_{10}, x_{11}, x_{12} ... x_{1n-1})$

$\vdots$

$X_{m-1} = (x_{(m-1)0}, x_{(m-1)1}, x_{(m-1)2} ... x_{(m-1)n-1})$.

$(x_{0j}, x_{1j}, x_{2j}, x_{3j} ... x_{(m-1)j})$, *where j = 0 ...n-1,* is called the observation column and all the values in the column have the same time stamp.

Definition 2: *Symbolic representation of a time series* X_i *is a mapping of real values of* X_i *into a symbolic string* $S_i = s_{i0}s_{i1}s_{i2}...s_{in}$ *such that* $s_{ij} = a_{il}$ *where* $l = Quant(X_i, x_{ij}, \mid \Sigma_i \mid)$ i.e a_{il} is the l^{th} alphabet of Σ_i.*Quant*$(X_i, x_{ij}, \mid \Sigma_i \mid$ is a *discretization function* which maps real value x_{ij} into positive integer value l, $0 \leq l \leq \mid \Sigma_i \mid -1$.$\mid \Sigma_i \mid$ is the size of the alphabet set Σ_i.

Definition 3: *A pattern p of length w<n is a contiguous substring* $(s_{ji}, s_{ji+1} ... s_{ji+w-1})$ *of string* S_j *where* $i+w-1 \leq n-1$ *and* $1 \leq i \leq n-w+1$.

A pattern p is called frequent pattern if the number of occurrences of p in $S_i \geq MIN\_SUPPORT$. In the following discussion we use the term $Pattern$ to refer to a frequently occurring substring.

Definition 4: *Let* $C = \{p_0, p_1, p_2 ... p_{max}\}$ *be the non empty set of all the patterns(substrings) from a sequence S. Clustering is partitioning of C into subsets* $C_0, C_1, ... C_{m'-1}$ *such that :*

1. $C_z \neq \emptyset$, z = 0...$m' - 1$
2. $\bigcup_{z=0}^{m'-1} C_z$ = C
3. $C_y \bigcap C_z = \emptyset; y \neq z; y, z = 0, 1 ... m' - 1$

Partitioning of the set C is done by a *clustering algorithm* (VI) .

Definition 5: *A* ***Position pair*** *is an ordered pair* (p_s, p_e) *where* $p_s \leq p_e$ *and* p_s, p_e *are both positive integers.* A position pair is used to represent the location and duration of a single occurrence of a pattern in a given sequence. The number p_s, called the *starting position* denotes the starting time of the pattern p in S_i, and p_e, called the *ending position,* denotes the ending time of the pattern p in S_i. For example, pattern 'abc' has a position pair (2,4) in the sequence 'efabcdef'. Two position pairs *A,B* such that $A = (a_s, a_e), B = (b_s, b_e)$ and $a_s \leq b_s$ *overlap* each other iff $b_s \leq a_e$. Two *overlapping positions pairs A,B* are said to be *merged* to form a *single position* B' *such that* B' = $(a_s, max(a_e, b_e))$. For example, in the sequence "ababa" the position pairs for "aba" (0,2) and (2,4) overlap and the merged position pair for (0,2), (2,4) is

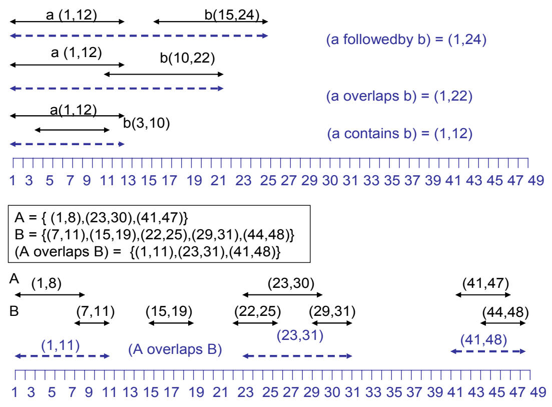


Fig. 3. Temporal Relation between two position pairs a,b and Position Lists A,B

given by (0,4).

We now define a list of positions at which different occurrences of a pattern appear in a sequence S_i. A *cluster position list* is then created by merging the position lists of all the patterns that belong to the same equivalence class.

Definition 6: *A **Position List** $L = \{(a_{s1}, a_{e1}), (a_{s2}, a_{e2}), ...(a_{sk}, a_{ek})\}$ is an ordered list of position pairs sorted by starting positions such that any two overlapping position pairs are merged until no overlapping positions are left.* A Position list has two important properties

1. All the position pairs are sorted by their starting position. i.e $a_{s1} < a_{s2} ... < a_{sk}$

2. No two position pairs overlap. i.e $a_{e1} < a_{s2}, a_{e2} < a_{s3}, ..., a_{ek-1} < a_{sk}$

A Position list for a pattern *p* in a sequence S_i is the ordered list of position pairs for all occurrences of *p*. This is obtained by merging all overlapping occurrences (position pairs) of p in S_i until no overlapping position pairs are found. For example, position list for pattern 'aba' in sequence 'abababcdaaba' is given by {(0,4),(8,10)}, here the position pairs (0,2),(2,4) are merged to (0,4). The idea of position list is extended to equivalence classes also. Since each equivalence class is a set of patterns, a position list for an equivalence class is defined as the *union of all position lists corresponding to the patterns contained in the equivalence class, such that overlapping positions are repeatedly merged to form a non overlapping sorted position-pair list*. We call the position list for an equivalence class as a cluster position list.

Definition 7: *A **Temporal Relation Function** $Rel \in \{overlaps, followedby, contains\}$ is a binary function defined for position pairs and position lists as follows.*

Let A (a_s, a_e), B (b_s, b_e) where $a_s \leq b_s$ be two position pairs and WIN be a user defined window size, then relation Rel(A,B), Rel $\in$ { overlaps, followedby, contains } is defined as

followedby(A,B) is true $\iff$ $(a_e < b_s) \wedge (b_s - a_e \leq WIN)$. Consequently, (A followedby B) is a new position pair given by (a_s, b_e).

contains(A,B) is true $\iff$ $(a_s < b_s) \wedge (b_e < a_e)$. Consequently, (A contains B) is a new position pair given by (a_s, a_e).

overlaps(A,B) is true $\iff$ $(b_s \leq a_e) \wedge (a_e \leq b_e)$. Consequently, (A overlaps B) is a new position pair given by (a_s, b_e). This definition of *overlap* contains Allen's [5] definition of *meets, starts, equals and overlaps*. By combining the meaning of meets, starts, equals and overlaps into a single definition of overlap, significantly reduces the number of temporal association rules in the search space. See figure 3 for examples of temporal relations *overlap, contains, followedby* between two position pair A and B. For example,with position pairs A(1,12) and B(15,24) and window WIN=5, we can see that *Followedby(A,B) is* **true** and new position pair *(A followedby B)* = (1,24) is created. Similar position pairs are created for other relations *overlaps* and *contains*.

The concept of temporal relations is extended to position lists from position pairs. Temporal relations between position pairs is defined in terms of position pairs, i.e if certain percentage(this is also called support threshold min_support) of time points in the time series hold the temporal dependency then the position lists are said to hold the temporal dependency.

Let L_1, L_2 be two position lists and Rel $\in \{overlaps, followedby, contains\}$ be a binary function on L_1, L_2. A new position list (L_1 Rel L_2) is created as (L_1 Rel L_2) = $\bigcup$ $(a\ Rel\ b)$, $a \in \ell$,$b \in L_2$, $\ell = \{a \in L_1 \mid Rel(a, b)\ is$ **true**$\}$.

$Rel(L_1$,$L_2)$ is defined as **true** iff $\frac{|(L_1\ Rel\ L_2)|}{p_{max}} \geq$ *min_support* else $Rel(L_1,\ L_2)$ is **false**.

The position list $(L_1 Rel\ L_2)$ is constructed by union (with repeated merging of overlapping position pairs) of all the position pairs $(aRelb)$,$a \in Ł_1$,$b \in L_2$ which satisfy $Rel(a, b)$. And for the Relation $Rel(L_1, L_2)$ to be true, the length of position list $(L_1 Rel L_2)$ be at least min_support times the length of the time series.

Definition 8: *A **Temporal Association Rule** between two Position lists is recursively defined as follows. This definition is based on* [1].

1. if X is a Position list then it is a temporal association rule. It is also called an *Atomic Temporal Association Rule*

2. if $Rel(X, Y)$ is **true** and X,Y are temporal association rules then $(X\ Rel\ Y)$ is also a temporal association rule.

The size of a temporal association rule is the number of atomic temporal association rules present in it. A rule of size k is called k-rule.

Problem Definition: The problem is to find large k - rules given the parameters *min_support, WIN,min_confidence* and the symbolic representation S_i for each feature/dimension in the multivariate time series.

IV. Symbolic Representation

Most of the times, it is necessary for the time series to be preprocessed before we could arrive at the symbolic representation. The sequence of tasks generally required, but not limited to, are 1. smoothing 2. normalization 3. dimensionality

TABLE I

EXAMPLE FOR TERMINOLOGY

Time Series	X_1 = 1,2,4.1,3.2,5.4,2.2, 4.6,1.2,2.4,4.3,5.3 3,5.1,1.2,2,6,3,5
time series sequence	S_1 = abdcebdabdeceabfce
frequent patterns	C = { bd,abd,ce,ab }
position pairs for 'ab'	(0,1),(7,8),(13,14)
position list	p_{ab}=$\{(0,1),(7,8),(13,14)\}$ p_{abd}=$\{(0,2),(7,9)\}$ p_{ce}=$\{(3,4),(11,12),(16,17)\}$
clusters	C_1 = {ab,abd } C_2 = { bd} C_3 = { ce }
cluster position list	P_{c_1}=$\{(0,2),(7,9),(13,14)\}$ P_{c_2}=$\{(1,2),(5,6),(8,9)\}$ P_{c_3}=$\{(3,4),(11,12),(16,17)\}$
temporal association rule;	C_1*followedby* C_3 min_support=2;WIN=3;
position List (C_1*followedby*C_3)	$\{(0,4),(7,12),(13,17)\}$

reduction 4. discretization 5. symbolic representation. The choice of algorithms in the tasks 1-5 are generally dependent upon the user requirements and the set of features which the pattern needs to capture. [11] provides an extensive survey of discretization methods. For example, should we wish to mine the trends instead of patterns in Time Series we can choose a method similar to [3]. Sometimes the time series is first differentiated with respect to time and then frequent patterns are found. Such time series identify the similar patterns occurring at different offset values. Discretization causes some loss of original information but a later step in our process clusters similar strings together and considers them as one equivalence class. This latter step significantly neutralizes the loss of information.

For our experiments smoothing was done using moving average smoothing and mean shift algorithm. Normalization was done using zscore and Min-max normalization. Dimensionality reduction and symbolic conversion can be achieved through PAA and SAX [15]. For discretization we have tried multiple algorithms such as equal-frequency discretization, equal-interval discretization and domain-expert directed discretization using a histogram. Our methodology is not dependent on any particular algorithmic implementation of the above 1-5 tasks.

V. Finding Frequent Patterns

After the time series is converted into a string our subproblem is to find the frequent substrings in the string using suffix trees. In a suffix tree for the given string, the number of times a particular substring/pattern occurs equals the total number of leaf nodes it has under it. The locations of each of these occurrences are traced back by following the pointers to the string from each of the leaf node. For example in figure 4 the pattern ***ce*** repeats 3 times and it has 3 leaf nodes. One of the challenges involved in such enumeration is the huge amount of data produced for keeping track of every pattern and all the locations at which the pattern occurs. A naive enumeration of

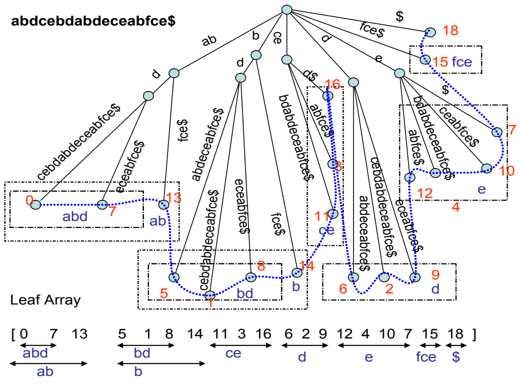


Fig. 4. Enhanced Suffix Tree for abdcebdabdeceabfce$

patterns and all their locations takes a lot of space to store all the locations of all the frequent patterns. To solve this, we use an efficient encoding scheme for storing all the locations for all the patterns. This encoding scheme preserves the linear time computational complexity of the algorithm while enumerating all the patterns and corresponding locations of each pattern. *It is space efficient and takes space equivalent to the total number of leaf nodes in the tree to store all the locations of all the frequent patterns.*

In a enhanced suffix tree, for each repeating pattern, all the corresponding locations are specified by only two parameters, *the index to leaf node* and *offset i.e. no of occurrences* in the *Leaf Array* L. We can store and enumerate all the locations of all the patterns by storing only Leaf Array and two parameters for each pattern. As an example, let us consider the pattern **bd** in Figure 4 . **bd** occurs at 3 locations (since node bd has 3 children) starting at positions 7,3,10. These occurrences can be represented just by specifying an index=3 and offset=3 on leaf array. The locations are given by L(3),L(4),L(5) which are 5,1,8. Now let us consider pattern **b**,the index of **b** is 3 and child_count is 4. So **b** occurs at L(3),L(4),L(5),L(6) which happens to be 5,1,8,14. We have implemented a modified ukkonen's algorithm [13] for suffix tree construction. The details of enhanced suffix tree construction, leaf array are omitted here for brevity.

This compact representation also enables pruning of redundant patterns. With the information stored in the suffix tree we can prune patterns based on 1. support, i.e. no of occurrences of the pattern (MIN_SUPPORT) 2. length of the pattern (MIN_LENGTH and MAX_LENGTH). 3. Removing redundant patterns.

Two patterns p_1, p_2 are redundant if p_1 has a position list = $\{(a_0, b_0), (a_1, b_1), ...(a_k, b_k)\}$ (where $a_0 \geq 1$, $b_k \leq n-1$) and p_2 has a position list $\{(a_0+1, b_0+1)$, (a_1+1, b_1+1), ...$(a_k+1, b_k+1)\}$ or $\{(a_0-1, b_0-1)$, (a_1-1, b_1-1), ...$(a_k-1, b_k-1)\}$. As an example, in figure 5 consider three frequent patterns in a sine wave p_1 , p_2 and p_3, these patterns are redundant because they always occur next to each other, and knowing the position list of one pattern we can calculate

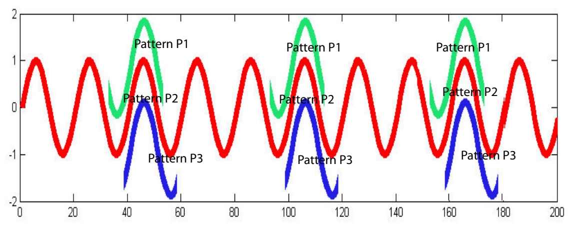


Fig. 5. Example of redundant patterns

the occurrences of other patterns. We reduce the number of candidate patterns by selecting only one representative pattern instead of the 3 redundant patterns. There can be many ways (or multiple bias) for choosing a representative pattern among the redundant patterns. We have experimented with various methods and finally implemented a method, which chooses the pattern with highest entropy as the representative and prunes out the rest of the redundant patterns. Thus we are keeping only those patterns which have high information and pruning out the redundant patterns with low information. Pruning of redundant patterns also avoids the problems which is common to subsequence clustering [12]. We have tested with multiple datasets and observed that when we do redundant pattern pruning, and pruning with frequency the mean of all the resulting subsequences is not a constant or a straight line. So our methodology does not have the drawbacks mentioned in [12], which normal subsequence clustering seems to have. Pruning also helps in drastically reducing the number of candidate patterns for the clustering step. The leaf array (see figure 4) contains the location information for each pattern and any two patterns can be checked for redundancy by looking at their respective index and offset on leaf array. The complete details of redundant pattern pruning algorithm has been omitted due to lack of space. Finally, the computational complexity of the algorithm to find frequent patterns and all their locations is linear in the length of time series since suffix tree construction [13] has linear time complexity (linear in the length of time series).However, redundant pattern pruning is not linear in the length of time series.

VI. Clustering

After obtaining all the frequent patterns our next goal is to find the classes of similar frequent patterns. This is accomplished by the clustering algorithm which outputs equivalence classes for the given set of frequent patterns and similarity measure. The similarity measure captures and quantifies the noise in the data. There are many symbolic similarity measures [4] which capture this kind of noise such as edit distance, longest common subsequence(LCS). The similarity measure which we have implemented for clustering the strings into equivalence classes, is based upon longest common subsequence. The advantage of such measure is that it allows for gaps in between the symbols of a pattern while matching them for similarity. Formally if s_1 , s_2 are two strings then similarity measure:

Sim(s_1, s_2) = $\frac{2*LCS(s_1,s_2)}{(|s_1|+|s_2|)}$ and

Dist(s_1, s_2) = 1 - *Sim*(s_1,s_2), is the distance measure. $LCS(s_1, s_2)$ is the length of longest common subsequence i.e. common subsequence in s_1, s_2 of maximal length. $| s |$ is the length of string s. The advantage of choosing such a non-metric similarity measure over the metric such as edit distance or Levenshtein Distance is that this similarity measure scales with the length of the strings unlike LCS which is an absolute measureand will not work for strings of which have a large variance in their lengths. Instead of string similarity measure, we also tried a numerical similarity measure (LCSS) among the numerical subsequences in original time series which correspond string subsequences found. LCCS was used by Vlachos *et. al*) [19] for clustering multi-dimensional trajectories. We have found that LCSS produces more meaningful clusters than LCS. Details of LCSS can be found in [19]. Using one of the above similarity measures we cluster the patterns in equivalence classes, However, for large datasets the number of patterns for each dimension can be very large, doing a hierarchical clustering is computationally expensive. So for large datasets, we use two-threshold Sequential Algorithmic Scheme [18] which has lower computational complexity. For smaller datasets we have implemented hierarchical clustering (Average Linkage Clustering) so that the user can can control the number of clusters based on the dendrogram visualization.

The details of the two-threshold Sequential Algorithmic Scheme can be found in [18]. Although any kind of clustering algorithm can be used in step 4, the choice of TTSAS for large datasets is justified by lower complexity and lesser dependence upon the order in which the values are input. Similarly, Hierarchical clustering for smaller datasets is justified by user interaction and control over the clustering.

VII. Temporal Association Rules

The clustering phase groups all similar patterns into an equivalence class. Now the objective is to find temporal relations between the clusters which we call as *Temporal Association Rules*. The Position List for each of the clusters is constructed as follows

```
for each Cluster $C_i$
    for each position $p_j \in C_i$ {
        Create a Position List $L_{pj}$ from Leaf Array
        $L_{Ci} = L_{pj} \vee L_{Ci}$
    }
```

The relationship between two Cluster Position List can be of any one of the {*followed by, Overlaps, Contains*} shown in figure 3. Moreover the three relationships mentioned above conceptually captures most of the temporal relationships possible and the commonly mined relationships. The first step is, start with atomic rules and then incrementally mine for higher order rules in a level wise algorithm. This means discovery of level 2 rules, followed by level 3, followed by level 4 until the maximum level specified is reached.

Mining for level 2-rules is done by doing all-to-all comparison of all the cluster position lists. Let $\mathcal{C}$ be the set of all the cluster position lists, *min_support* be minimum value of support and *min_confidence* be minimum value of confidence. $\mathcal{C}_2$ be the set of 2-rules initialized to $\emptyset$

```
1:  level = 1
2:  $\mathcal{C}_{level} = C$
3:  while $level + 1 < max\_level$ do
4:    for each cluster $c_i$ in $\mathcal{C}_{level}$ do
5:      for each cluster in $c_j$ in $\mathcal{C}$ do
6:        for each relation rel $\in${followed by, Overlaps, Contains} do
7:          if ( $\frac{Card(c_i)}{Card(c_i\ Rel\ c_j)} > min\_confidence$) then
8:            $\mathcal{C}_{level+1} = \mathcal{C}_{level+1} \bigcup (c_i\ rel\ c_j)$ {Add the newly generated position list to candidates for next level}
9:            Compute the Position List for $c_i\ rel\ c_j$
10:         end if
11:       end for
12:     end for
13:   end for
14:   level = level + 1
15: end while
```

The confidence of $(c_i\ Rel\ c_j)$ is defined as $\frac{|\bigcup a_i|}{|c_i|}$ where a_i is a position pair, $(a_i \in c_i) \bigwedge (Rel(a_i, c_j)\ is\ true)$ i.e the number of position pairs in c_i for which Rel(c_i, c_j) is true divided by the total number number of position pairs in c_i . This kind of rules are similar to A-1 type of mentioned in [1]. After finding all the frequent k-rules rules are pruned out for high confidence rules. We can also modify the algorithm to output any level-rule if the confidence is greater than some threshold. While confidence is one way to find relevant and important rules Summarization is another way to accomplish this task.

VIII. Summarization

Summarization is the process which generalizes the temporal association rules by identifying the time windows in which the rule is applicable. Summarization creates a new position list by combining all the position pairs of rule, which are closer than window SUMWIN. The output is a new position list containing each stretch of temporal dependency as position pair. The measures for summarization are coverage, average length of coverage and maximum length coverage. Coverage measures the total span of a summarized rule in the time series. Average length of coverage measures the average length of the stretch of the summarized rule. It tells whether the dependency holds contiguously as a long stretches (for example seasonal patterns, modes of operation, events) or whether it holds in discrete short stretches. Maximum length Coverage measures the longest stretch of occurrence of the temporal dependency, sometimes indicating an important event or condition. The exact interpretation of the average length of coverage and maximum length coverage are domain dependent. For example, figure 6(a) shows a temporal dependency in a single dimension where high energy consumption in day is followed by low consumption at evenings and nights. We can see that this dependency holds during weekdays only and not on weekends. Summarized rule in figure 6(b) identifies the stretches or the weeks where the dependency holds. So by looking at the summarized rule we can figure out that the rule is applicable only on weekdays and not on weekends and holidays.

Definition 9: Let P_{ij} be position list corresponding to $c_i\ rel\ c_j$ then $Summ(P_{ij}, SUMWIN)$ is a summarized position list corresponding to $c_i\ rel\ c_j$ and window length SUMWIN such that for any two position pairs a,b in P_{ij} where $a < b$ and b_s - a_e $\leq$ SUMWIN are merged to form a single position list c = (a_s, b_e) until there are no further position pairs a,b with b_s - a_e $\leq$ SUMWIN. Below is the algorithm to compute the Summarized position list from the position list of a rule and summarization window.

```
{ This algorithm takes a position list p corresponding to a rule, summarization window SUMWIN and outputs summarized position list $s$}
i = 0, j = 0, $s_j.start$ = $p_i.start$
while $i <| p |$ do
  if $p_{i+1}.start$ - $p_i.end >$ SUMWIN then
    $s_j.end$ = $p_i.end$
    if $i <| p |$ then
      $j = j + 1$
      $s_j.start$ = $p_{i+1}.start$
    end if
  end if
  $i = i + 1$
end while
$s_j.end$ = $p_i.end$
```

Summarization finds the regions where a rule is applicable and *measuring the temporal extent of these regions* would indicates how important a particular rule is. Coverage is one such measure which calculates the extent of the regions where the rule is applicable. Coverage is defined as,

Definition 10: Coverage measures percentage of the time points in the time series for which the summarized temporal rule applies. Let SP_{ij} be summarized position list corresponding to $c_i\ rel\ c_j$ and window SUMWIN, then coverage($c_i\ rel\ c_j$) is defined as

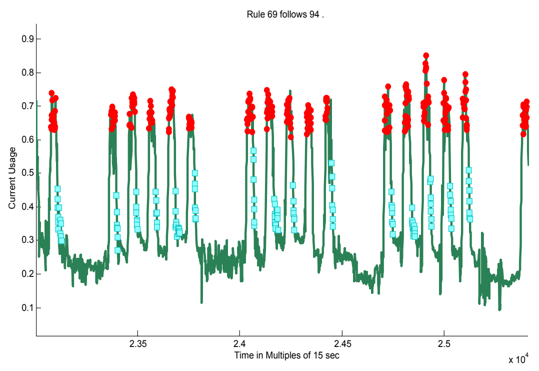
$$\frac{\sum p \in SP_{ij} \mid p \mid}{\mid p_{max} \mid}$$

where p_{max} is the position pair (0,n) and $\mid p_{max} \mid$ = n and $\mid p \mid$ for position pair $p = p_e - p_s + 1$. The average coverage length measures average length of the position pair in the summarized position list. Average coverage is given by

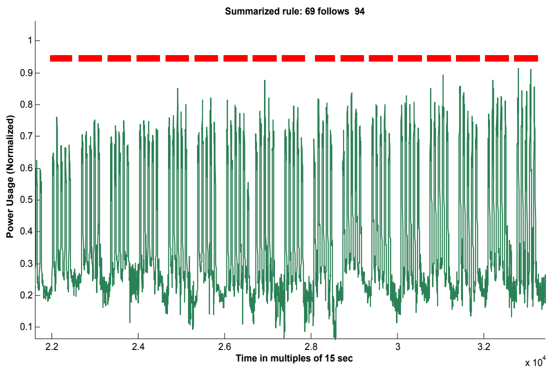
$$\frac{\sum p \in SP_{ij} \mid p \mid}{\mid SP_{ij} \mid}$$

. The maximum coverage is defined as the length of the longest position pair in the summarized position list SP_{ij} and it is given by $max(\mid p \mid)$ where $p \in SP_{ij}$.

For an example of summarization, we take the energy data set of industry having two variables, pressure and current of a



(a) Example of daily energy usage rule



(b) Example of Summarized daily energy usage rule

Fig. 6. Summarization

compressor. Each temporal rule maps to mode of operation of the compressor and the coverage tells us how long the compressor runs in a particular mode and when does the modes of operation change and this information is valuable to optimize the current consumption.

IX. Experimental Results

Experiments were ran on real data sets from various domains. We discuss the results from some of the datasets below.

Energy Dataset:

The experiments were done with a real large data set which recorded production parameters of an industrial compressor system. The attributes measured were current, pressure inside the compressor and discharge pressure at intervals of 10 seconds for 1 month, this constitutes 259200 observations.A sample rule denoting operating mode between current and pressure,(pattern #46 contains pattern #87) is shown in 7(a). A domain expert can use the values of average coverage and maximum coverage to identify the long operating modes and reduce the breaks in operating modes and thus cut energy consumption costs. After a review it has been found by domain expert that our methodology does identifies the actual operating modes.

Other Datasets:

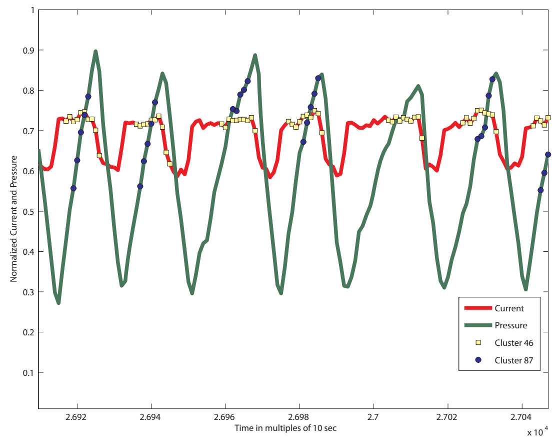
A sample of EEG data set from [9] has been tested to see the scalability of our methodology. The dataset contains 1 million points each containing a single electrode voltage value. In a sample run there were 4935 frequent patterns and 120 clusters from the frequent patterns. The association rules searched were of *”followed by”* type since there was only one dimension. figure 7(f) shows the scalability of the mining process with the size of dataset. The algorithm has been run with a minimum pattern length of 4 characters and maximum length of rule of 4(4-rules). This shows that the algorithm can be run in reasonable time for time series lengths up to 1 million, after which the physical memory limitations seem to affect the performance.

We have also tested on the Great lakes dataset [9]which contains 5 dimensions with 984 data points, the 5 dimensions correspond to historical water levels of the following lakes: Erie, Huron, Ontario, St Clair and Superior. The records consist of data from 1918 - present recorded monthly. A sample rule found between the lakes Ontario, St Clair and Huron is shown in figure 7(d). The rules shows the seasonal variation of water level between the three lakes and the order in which the variation happens.

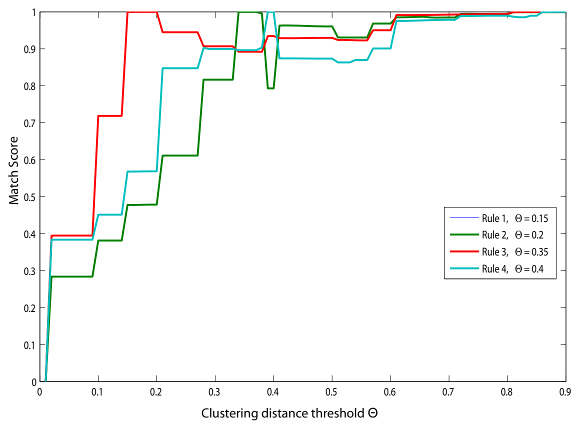
Robustness Analysis

A number of tests were conducted to observe the effect of clustering on the number of rules produced and the content of the rules themselves. Since we used the sequential clustering algorithm [18], we could control the clustering process by changing just one parameter called the Clustering distance threshold *distance threshold,* Θ, which is between 0 and 1. $\Theta = 0$ means that each pattern is an individual cluster, $\Theta = 1$ means that all the patterns are grouped into one single cluster forming a single equivalence class. As the value of Θ increases from 0 to 1 , the compactness of resultant clusters decreases.

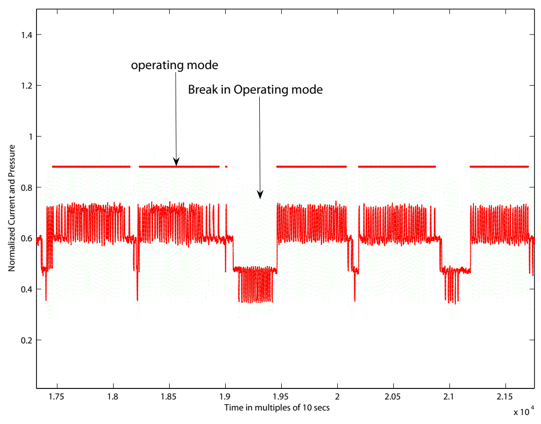
The tests were ran with the energy dataset with three dimensions (current, pressure and discharge pressure of a compressor) containing 20,000 points in the time series. Figure 7(e) shows the relation between the clustering distance threshold θ and the number of equivalence classes produced. We can observe that as Θ gets closer to 0, the number of clusters increases almost exponentially. As Θ is increased the number of clusters, and the number of rules produced are exponentially reduced demonstrating that clustering does control the exponential explosion of frequent patterns. Also, the number of clusters do not change for various neighborhoods of θ values. We believe this is due to string similarity measures in



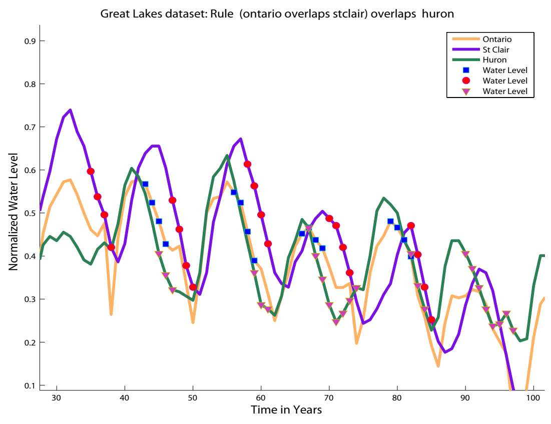
(a) rule =(46 contains 87), support = 13.1confidence = 0.9



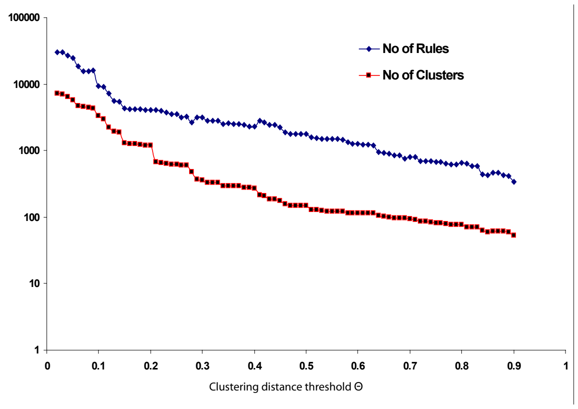
(b) Effect of clustering threshold Θ on robustness of the rules



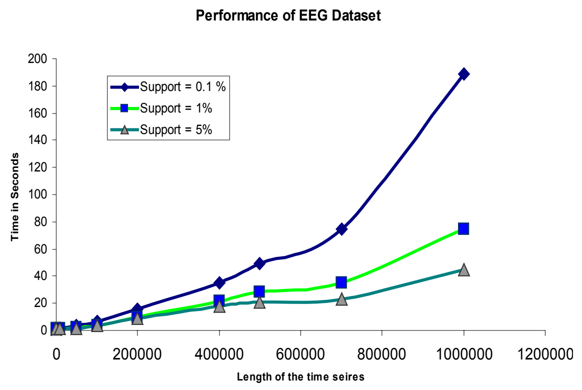
(c) Summarization of rule 46 contains 87



(d) Great Lakes dataset, Rule (Ontario overlaps StClair) overlaps Huron



(e) Effect of Clustering



(f) Scalability with EEG Dataset

Fig. 7. Temporal Association Rules

clustering, resulting in multiple bands of stable regions where changing the Θ does not drastically change the number rules and equivalence classes.

We have tested the extent of similarity among the rules produced with different values of clustering threshold Θ. That is, if a rule is produced with a particular value of Θ, say Θ_1, then the same rule should be preserved in a different set of rules produced with a different value of different value of Θ. To compute the match, we take a particular rule r_1 produced with Θ_1 and look for the best rule r_2, which matches r_1 among the set of rules produced with a different value of Θ say Θ_2 . The quantitative measure of match between two rules r_1, r_2 is

$$Match(r_1, r_2) = \frac{BitVec(r_1).BitVec(r_2)}{BitVec(r_1).BitVec(r_1)}.$$

The BitVec(**r**) of a rule is **r**, is a vector of length n (n=total length of time series) containing 0's and 1's. A value of 1 is assigned for the locations where the rule **r** holds and 0 where the rule **r** does not hold in the time series. $Match(r_1, r_2)$ is is the dot product of corresponding bit vectors of r_1 *and* r_2 normalized by dividing with number of 1's in r_1. $Match(r_1, r_2)$ is a score between 0 and 1, 0 being there is no match and 1 being a complete match. We have picked a few rules randomly produced with different values of Θ and checked how they are preserved in the sets of rules produced with varying Θ. Figure 7(b) shows the results of this comparison. We can see that the rules, match decreases as the Θ decreases. This is expected due to the fact that as Θ decreases the number of clusters increases, compactness of each cluster increases, and the chance of clusters matching across the rules decreases. We can see that the rules are preserved across the whole range of Θ except for very low values of Θ (unstable region). More importantly, The rules do not change drastically with a variation in Θ and there are stable regions where clustering does not effect the number of rules produced and also the content of the rules. So we can say that our methodology does produce meaningful rules and stable rules and these rules are robust against small variations in clustering parameters. Furthermore, as shown in figure 7(e), we control the exponential explosion of frequent patterns through clustering to produce equivalence classes and thus meaningful and relevant rules.

X. Conclusion

We have presented a methodology for mining temporal associations among frequent patterns occurring in multi-variate time series data. This methodology seeks to control the exponential explosion of strings by clustering similar strings into equivalence classes. We also seek to discover temporal associations among these classes from a richer set of possibilities. We have shown that this algorithm yields meaningful relationships in real-life data sets. The algorithm is also scalable and can be applied to large datasets. The summarization aspect of our methodology identifies time periods during which a particular temporal dependency holds. This is equivalent to identifying modes of operation of a system. We have shown that we can discover patterns at various levels of abstraction, starting from a very fine granularity data and in a scalable manner. These capabilities give a user more power than any other framework presented in the literature.

Acknowledgment

The authors would like to thank Chandan Rao from Graphet Inc for providing the compressed air datasets and domain expertise.

References

[1] Po-Shan Kam and Ada Wai-Chee Fu. *Discovering temporal patterns for interval-based events*. In Yahiko Kambayashi, Mukesh K. Mohania, and A. Min Tjoa, editors, *Second International Conference on Data Warehousing and Knowledge Discovery* (DaWaK 2000), volume 1874, pages 317326, London, UK, 2000. Springer.

[2] Gautam Das, King-Ip Lin, Heikki Mannila, Gopal Renganathan, Padhraic Smyth, *Rule Discovery from Time Series*, Proc. Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), New York, New York, (August 27-31, 1998), pp. 16-22.

[3] Udechukwu,A., Barker, K. Alhajj, R. (2004). *Discovering All Frequent Trends in Time Series.*In proceedings of the 2004 Winter International Symposium on Information and Communication Technologies (WISICT 2004).Jan 5-8. Cancun, Mexico

[4] Dimitrios Gunopulos, Gautam Das: *Time Series Similarity Measures and Time Series Indexing. SIGMOD Conference 2001*

[5] J.F. Allen. *Maintaining knowledge about temporal intervals*. Communications eed of the ACM, 26(11):832–843, 1983.

[6] Villafane, R., Hua, K.A., Tran, D., Maulik, B.: *Mining interval time series*. In: Data Warehousing and Knowledge Discovery. (1999) 318-330

[7] J. Lin, E. Keogh, P. Patel, and S. Lonardi. *Finding Motifs in Time Series* . In proceedings of the 2nd Workshop on Temporal Data Mining, at the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Edmonton, Alberta, Canada. July 23-26, 2002.

[8] Höppner, F. (2001). *Discovery of temporal patterns learning rules about the qualitative behavior of time series*. In Proceedings of the 5th European Conference on Principles and Practice of Knowledge Discovery in Databases. Freiburg, Germany, pp 192-203.

[9] E. Keogh, *The UCR time series data mining archive,* http://www.cs.ucr.edu/ eamonn/TSDMA/index.html, University of California Computer Science and Engineering Department, Riverside, CA, 2003.

[10] C. Mooney, J.F. Roddick. *Mining Relationships between Interacting Episodes. In Proc. 2004 SIAM International Conference on Data Mining*, Orlando, Florida. Dayal, U. and Berry, M. W., Eds.

[11] Daw CS, Finney CEA, Tracy ER (2003). *A review of symbolic analysis of experimental data*. Review of Scientific Instruments 74: 916-930.

[12] Keogh, J. Lin, and W. Truppel. (2003). *Clustering of Time Series Subsequences is Meaningless: Implications for Past and Future Research*. In proceedings of the 3rd IEEE International Conference on Data Mining . Melbourne, FL. Nov 19-22. pp 115-122

[13] E. Ukkonen. *On-line Construction of Suffix Trees.* Algorithmica, 14(3) pp249-260, 1995.

[14] Mörchen, F., Ultsch, A.: *Discovering Temporal Knowledge in Multivariate Time Series*, Proc. GfKl Dortmund, Germany, pp272-279, 2004

[15] Lin, J., Keogh, E., Lonardi, S. & Chiu, B. (2003) *A Symbolic Representation of Time Series, with Implications for Streaming Algorithms*. In proc. 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery. pp2-11, San Diego, CA.

[16] Chan, K. & Fu, A. W. (1999). *Efficient time series matching by wavelets*. In proceedings of the 15th IEEE Int'l Conference on Data Engineering. Sydney, Australia, Mar 23-26. pp 126-133.

[17] B. Chiu, E. Keogh, and S. Lonardi. *Probabilistic discovery of time series motifs*. In Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, Washington, 2003, pp 493-498.

[18] S. Theodoridis.,K. Koutroumbas, Pattern Recognition, Academic Press, 1998, pp 438-440

[19] M. Vlachos, G. Kollios, ans G. Gunopulos, *Discovering Similar Multidimensional Trajectories*, in Proc. of the 18th ICDE, San Jose, CA, 2002, pp. 673684.