Mining Temporal Databases for Subsequence Patterns

Wen Niu, Raj Bhatnagar
ECECS Department, University of Cincinnati, Cincinnati, OH 45221, USA
wniu@ececs.uc.edu, Raj.Bhatnagar@uc.edu

Abstract

We consider the problem of mining databases containing time series data for discovering typical temporal patterns. A gene micro-array database may contains temporal expression profiles of thousands of genes and it would be insightful to determine time intervals during which a number of genes share their expression levels. In another example temporal profiles of electricity consumption during a day are recorded and such records for a number of years may be stored in a database. Our goal is to determine typical temporal behaviors of utility consumption and the days on which such behaviors are expressed. An interesting temporal pattern is a subsequence of observed values that is shared by profiles in a reasonably large number of records.

1 Introduction

We view each temporal profile as a sequence of observations at successive points in time. A profile may represent expression levels of a gene measured at different time points during a biological process or it may be a record of electric power consumption measured at 15-minute intervals in a day. Our objective is to discover hypotheses of typical temporal behaviors and the identities of profiles that exhibit such behavior patterns. An interesting pattern of temporal behavior, for our methodology, is any subsequence of the complete temporal profile that is shared by a reasonably large number of profiles. For example, let us consider temporal profiles: abacbde, adabbde, aaaabde, baacbde, and acadbde. Each profile represents observations at seven successive time points. The subsequence, "a-a-bde" is shared by profile # 1, 2, 3, and 5. The subsequence "—-bde" is shared by all the five profiles.

2 Related Work

In the context of temporal gene data a number of methodologies have been employed to cluster them. These results are presented in [Eise 98, Holt 00, Spel 98]. All these methods employ traditional clustering algorithms and use Euclidean distances or various correlation coefficients as metrics of distance between temporal profiles. When a pattern is defined by a subsequence, it is not possible to compute a distance metric unless the subsequence is identified. These distances are used to drive the clustering algorithms [Eise 98] but are not available when the distance between profiles depends on yet to be identified subsequences. The approaches presented in [Agra 93, Spel 98, Holt 00] assume an underlying cyclical characteristic and cluster profiles based on their similarity in the Fourier or other transformed characteristic space. It is difficult for these approaches represent subsequence type of temporal concepts.

3 Methodology

In abstraction, we view our temporal data as a table in which each row is a temporal profile and each column corresponds to a point in time at which the process is observed. The algorithm reads records from the database one at a time and considers each record as providing one row of this table. An interesting temporal hypothesis is characterized by a subsequence that is shared by a number of rows in the table. Intuitively, the preference is to find *larger* subsequences, each of which is shared by a *larger* number of rows. We refer to each element of this table as $D(p_i, t_j)$ where p_i is the profile (row) id and t_j is the j^{th} observed time point.

3.1 A Temporal Hypothesis A temporal hypothesis consists of a *subsequence* and a set of profiles that contain it. We define a temporal hypothesis h to be:

1. A subsequence T containing an arbitrary choice of columns in D. The these m distinct column numbers (m time points) are: $(v_1, v_2, \ldots, v_m)$.
2. A set R containing k distinct rows: $(r_1, r_2, \ldots, r_k)$.
3. A metric to determine distance between two profiles (rows) r_1 and r_2 is defined as follows:
   $dist(r_1, r_2) = \sum_{v \in T} \mid D(r_1, v) - D(r_2, v) \mid$
   Intuitively, $dist$ is the sum of differences in values when only the time points contained in the subsequence T are considered. The rows contained in R are such that the distance $dist(r_j, r_k)$ between any r_j and r_k of R is below some threshold.

The goal of the mining algorithm is to identify all hypotheses h, given by pairs, $< T, R >$, such that they satisfy the distance metric constraint defined above. Identification of such temporal hypotheses can be modeled as a two-step clustering process. In the first clustering task we identify those **contiguous substrings** within the profiles whose shared values occur frequently in many rows. The second clustering task is to identify sets of rows Rs that share a number of contiguous substrings identified in the first task. Considering the example of section 1 above, a at location 1 is a substring shared by profile #s 1, 2, 3, and 5; The substring "a" at location 3 is shared by all the five profiles, and substring "bde" starting at location 5 is shared by all the five profiles. The second clustering task combines the rows that share a number of discovered substrings.

3.2 Algorithm Outline We first quantize the numeric values in the table D into a small set of value-ranges and represent each value-range with a symbol (an alphabet). In the examples considered here we have used symbols *d, c, b, a, *, A, B, C, D* to represent the value ranges of monotonically increasing numeric values such that an *'*'* represents a zero or a near zero value, A very small example to illustrate our methodology is given below and it consists of the following four strings: *ABacxBC, ABCD*bd, ABaaxbd, x*a**bd* These four strings represent four temporal profiles containing observations at seven time points each. The task of identifying interesting temporal hypotheses is completed in the following three phases:

1. **Phase-1:** Determine the set S of all shared *contiguous substrings* such that:
   (a) Each substring $s \in S$ occurs in a set of strings corresponding to profile (row) ids $G = (g_1, g_2, \ldots, g_k)$ and in each complete profile g_i the substring s starts at the same location (fixed time point).
   (b) Pair $< G, s >$ represents an interesting (contiguous) substring s and the set of profiles G whose strings share this substring (beginning at some fixed time point)
2. **Phase-2:** Cluster the sets of profiles G derived in Phase-1 such that a cluster contains a number of those G sets that have many common g_is in them. The union of substrings corresponding to all G's included in such a cluster forms the temporal subsequence hypothesis characterizing the cluster and is called by us the *core* of the cluster.
3. **Phase-3:** Generalize (specialize) the core subsequences found in Phase-2 to search the space of temporal hypotheses for more interesting hypotheses.

3.3 Phase-1: Identify Shared Substrings A *Generalized Suffix Tree* (GST) is a tree of all suffixes in a set of given strings and is a well-studied data structure for a number of string matching applications [Gusf 97]. Efficient algorithms [Hui 00] for constructing suffix trees have been sufficiently investigated. For n input strings of m symbols each, the longest substrings shared by any k of the n (for all k between 2 and n) input strings can be found in $O(nm)$ time [Hui 00]. We have adapted the GST construction algorithm presented in [Doro 00] with the modification that in our tree, instead of suffixes, we maintain all substrings that start at a common starting location in a set of strings. This *GST* forms the basic knowledge structure for the phase-2 of our methodology.

3.3.1 GST Construction Algorithm We first consider all input strings and build a *GST* for suffixes starting at the first location (#0) of each string. This algorithm works on a concatenation of all strings, and uses the position numbers of substrings in this long concatenation. For the example mentioned above the concatenation of the four input strings is:

```
ABacxBC.ABCD*bd.ABaaxbd.x*a**bd.
```

While concatenating, an *end-of-string* marker character '.' is introduced at the end of each string. For the four original strings the *GST* generated, considering only the substrings starting at location #0 of each string is as follows:

```
(0)<root>-(8)AB-(10)CD*bd-(15)8
                -(18)a-(19)axbd-(23)16
                      -(3)cxBC-(7)0
        -(24)x*a**bd-(31)24
```

This *Tree* is a record of all shared substrings, their starting locations, and the original string numbers in which they occur. The internal node labeled *(19)axbd* implies that substring *axbd* occurs at location #19 in the concatenated string. Similarly, label *(18)a* implies occurrence of a at location 18 in the concatenated string. Multiple child nodes of a parent node signify continuation of the string represented at the parent node by different extensions in different strings. A leaf node label, for example, *(23)16*, signifies a string denoted by the labels on the path from the root node to this leaf node, which starts at location #16 and terminates at location #23 (of the concatenated string).

The tree contains information about all suffixes of the four profile strings in a very concise form. This tree can be read for two interesting suffix substrings as follows. The first node below the root node is marked $(8)AB$ and the second node below the root node is marked $(24)x * a * *bd$. This means the collection of strings has two kinds of strings, those that start with AB and those that start with $x * a * *bd$. The node

labeled (8)AB has two child nodes indicating that the string set starting with AB continues in two different ways, either with $CD*bd$ or with a. The two leaf nodes under the node marked (18)a indicate that there are two strings that share the suffix formed up to this node of the tree, that is, the suffix ABa. It can be seen that the count of leaf nodes under any internal node of the tree represents the count of strings that share the suffix formed between the root and the internal node. The information about two interesting substrings ABa and AB that can be read from this tree is listed in the table below. The columns in the table are: the substring, length of the substring, its starting position, set of string numbers in which the substring occurs, and the cardinality of this set.

```
ABa     3 0     {0,2}     2
AB      2 0     {0,1,2}   3
```

We then repeat the above process for the four strings, but with each string truncated by removing its first character. This, in effect, generates the GST for substrings originating at second position of each string (location # 1). The four substrings, now, are:

BacxBC BCD*bd Baaxbd *abd**

The suffix tree generated for the truncated strings is:

```
(0)<root>-(21)*a**bd-(27)21
          -(7)B-(8)CD*bd-(13)7
               -(15)a-(16)axbd-(20)14
                    -(2)cxBC-(6)0
```

The two substrings identified in this GST are:

```
Ba    2 1     {0,2}     2
B     1 1     {0,1,2}   3
```

We continue the rounds of GST construction by truncating the first character from every string until the strings of length one are examined. All the generalized suffix tree can now be viewed as child nodes of a super tree. The original GST construction algorithm had a complexity of $O(nm)$ [Hui 00]. Since we generate suffixes starting at every location of the string, the algorithm is repeated for all values of string length from 1 to m. The effective complexity of our algorithm, therefore, becomes $O(nm^2)$.

3.4 Phase-2: Merging Profile Sets Each substring identified in Phase-1 can be represented as a pair $< G, s >$ where G is a set of profiles (rows) $(g_1, g_2, \ldots g_m)$ in table $D(:,:)$ and each of these rows shares the common substring s, starting at some fixed common location in each string. In the second phase we cluster the sets Gs from among all the $< G, s >$ pairs generated in the first phase. We seek to identify those clusters of Gs that contain very similar row numbers. Similarity for a pair of Gs can be measured in a number of different ways and one such metric for two sets $G1$ and $G2$, that is used for the results of datasets in this paper, is: $\frac{Size(Intersection(G1,G2))}{Size(Union(G1,G2))}$. That is, if the cardinality of intersection of two sets is more than some threshold fraction of the cardinality of the union of the two sets, then we consider them to be reasonably similar.

Each node of the tree contains a G set and any two G's for which the above metric of similarity is above some threshold are merged together. Implementation of the above process can be viewed as a round of synthesizing and inheriting the attributes as follows.

1. Synthesis: Each internal node of the tree collects the G sets from all its children nodes into a set of $< G, s >$ pairs, let us call it $\mathcal{G}$, and the process moves up to the root node.
2. Inheritance: Each parent node passes its synthesized set $\mathcal{G}$ to all its children nodes. At each child node, each member of the local $\mathcal{G}$ and inherited $\mathcal{G}$ is compared to determine their closeness. If a pair of G's is found to be close, then a new hypothesis given by $< intersection(G_1, G_2),\ (s_1, s_2) >$ is added to the local $\mathcal{G}$. This process of comparing the members of local and inherited $\mathcal{G}$'s is continued until all close pairs have resulted into new hypotheses. Thus, hypotheses with multiple contiguous substrings are generated at each node of the tree.

A merger of the substrings (s of the merged $< G, s >$ pairs) is then performed, resulting in a subsequence hypothesis. The set of profiles corresponding to this merged substring are determined by the above process to be:

$R = rows.in(G_1, G_2, \ldots, G_k) = rows.in(G_1) \cap rows.in(G_2) \cap \ldots \cap rows.in(G_k)$

This ensures that the profiles in R certainly share all the substrings $(s_1, s_2, \ldots, s_k)$ which together form the subsequence T. The example below first shows the G sets at internal nodes that got created because of shared substrings and then the Table 3.2 shows the members of $\mathcal{G}$ that resulted by merging some G sets and combining their substrings into subsequences during the synthesis and inheritance steps of Phase-2.

Table 3.1: Strings After Phase-1

s	size(s)	set G	size(G)
ABa----	3	{0,2}	2
AB-----	2	{0,1,2}	3
--a----	1	{0,2,3}	3
----*bd	3	{1,3}	2
-----bd	2	{1,2,3}	3

Table 3.2 Clusters after Phase-2

T	size(T)	set R	size(R)
AB---bd	4	{1,2}	2
--a--bd	3	{2,3}	2

The first row of the above result means that the subsequence `AB---bd`, of length 4, occurs in rows numbered (1,2) and specifies an interesting temporal hypothesis. A rows in Table 3.2 is a temporal hypothesis that specifies a set of profiles R and a subsequence T across time. Some criterion of interestingness can be defined and evaluated for each temporal hypothesis and the Phase-2 processing can be made to output only the top some desirable hypotheses.

3.5 Phase-3: Generalization of Cores Each hypothesis in Table 3.2 specifies a core subsequence T that is shared by each profile of the set R. The cores and hypotheses generated by the two phases are affected by various factors such as the noise in the data and loss of information due to quantization of numeric values. We now seek to examine all interesting hypotheses in the vicinity of each seed hypothesis generated by the first two phases. The operations for searching this space modify the core subsequence by one of the following two basic operations:

Generalization: An alphabet in the core subsequence is allowed to be replaced by any one of a set of alphabets. For example, a B occurring in a subsequence may be allowed to be replaced by any member of the set A, B, C.

Specialization: A location not included in the core subsequence is instantiated to some particular subset of alphabets.

Generalizing a *core subsequence* has the effect of bringing in some additional profiles into a hypothesis. Specialization has the effect of reducing the number of profiles included in a hypothesis. A number of search strategies can be adopted to systematically search the space around each seed hypothesis and select the very good hypotheses. A core subsequence can also be written as a *regular expression*, or *regex*, in short. One of the many possible different ways the *regex* for a core may be generalized, and have been used in our tests is: Each symbol in a *regex* is considered replaceable by any of its immediately preceding or following alphabets. For example, in the core string BB–ab, we accept alphabets A, B, and C in place of each B, and $*$, a, or b for a and so on.

4 Test Datasets

We have tested our methodology with two different datasets. The **Yeast Cell-Cycle Dataset** presented in [Spel 98] represents expression levels of 6178 yeast genes from four time-course experiments. The dataset $D(:,:)$ is a table $D(1..6178, 1..77)$. Missing values cause problems in many clustering algorithms and should be dealt with appropriately. In our methodology these missing values do not hinder formation of temporal hypotheses if rest of the string, $D(i,:)$, shows sufficient similarity with a cluster. In the **Utility Dataset** each record contains 96 measurements of electric power (KWH) being consumed by an industrial plant. A measurement is recorded every 15 minutes and a profile contains all the measurements for one 24-hour period. Such records for each of the 365 days of a year are available.

5 Results with Cell Cycle Dataset

We selected a core hypothesis that happened to contain four histone genes (rows of the table D) and then generalized its core subsequence using a generalization rule. The core subsequence of this hypothesis is:

```
CC-b--c---------BCC-----C---bcc----C--
-----b----A-----b-----------b-B--B----b
```

To generalize this core subsequence, symbols at every position in all the 4 strings inspected. A position is labeled with a '+' if all symbols at this position are upper case letters, a '-' for all lower case letters, and an 'x' if the letters are of mixed cases. We allow a missing value to be belonging to the same *case* as that of all the other alphabets in this position, if they are of the same *case*. The following string representation is obtained after the symbols at every position are scanned:

```
++x----x+++x---x+++xxx--+++x----x+x+++x
x-x--xxxxx+x-x--x++x+x+x--x-x++++++xx-
```

In this example only those string positions are generalized in which a '+' is surrounded by a '+' on both sides and a '-' is surrounded by a '-' on both sides. After this generalization step the *regex* obtained is:

```
^.{4}[cdx]{2}.{3}[BCx]{1}.{3}[abx]{1}.{3}
[BCDx].{7}[BCx]{1}.{3}[bcdx]{2}.{5}[BCx]
{1}.{33}[BCx]{2}[ABCx]{1}[ABx]{1}.{3}$
```

and it is to be interpreted as follows: It matches any string that starts with any 4 characters, followed by any 2 characters from the character set [cdx], followed by any 3 characters, followed by any one from the set of [BCx], and so forth. When this generalized *regex* is searched against the entire set of symbolic expression profiles for 6178 yeast genes, a cluster of nine genes is found. This search against all strings is carried out against the trees that have been generated and the database does not need to be consulted again. A plots of the expression values for these nine genes is shown in Figure 1 below. All these 9 genes encode histone subunits, and they are involved in the same cellular process, with peak expression times in S phase of the cell cycle. Spellman *et. al.* [Spel 98] note that this is the most tightly formed cluster of any of the cell cycle genes. This example validates our methodology by demonstrating that temporal hypotheses learned correspond to known phenomena.

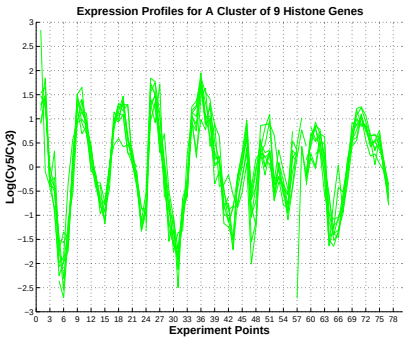

Figure 1: Expression levels of histone cluster

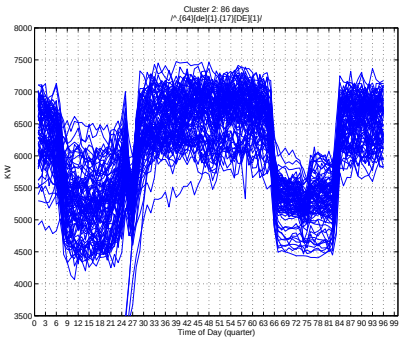

Figure 3: Hypothesis #2 about Utility Data

6 Results with Utility Data

We show two interesting hypotheses obtained after the generalization step was applied to the seed temporal hypotheses. These two clusters are shown in Figure 2 and Figure 3 below. The two dips around time point numbers 25 and 75 in each plot are around the start time of a shift in the morning and the shift shutoff time during the evening. The second dip around time point number 75 in Figure 5 takes a sharp $'V'$ kind of shape and in Figure 6 it takes a 'U' kind of shape. This is the main difference between the core subsequences of the two temporal hypotheses.

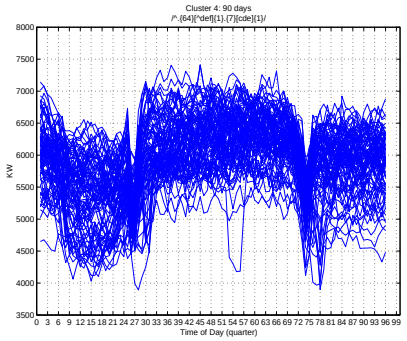

Figure 2: Hypothesis #1 about Utility Data

The temporal profiles in Figure 2 are the result of an inefficient shutdown procedure and the core subsequence in Figure 3 are the result of a proper shutdown procedure. An examination of the profiles included in each hypothesis reveals that all the days included in the hypothesis of Figure 2 are weekend days and all the days included in Figure 3 are the weekdays. The discrepancy is probably due to the fact that the weekend crew in the plant has not been following the proper shutdown procedure. That is of significant interest to the users.

7 Conclusions

We have presented a methodology for mining subsequence based temporal concepts from databases containing time series data in their records. Our approach considers each temporal subsequence to be a hypothesis about a characteristic temporal subsequence embedded in a set of profiles observed during some process. This capability provides significant power to an investigator looking for hypotheses about expressed temporal behaviors.

References

[Agra 93] Agrawal, R., Faloutsos, C., Swami, A. Efficient Similarity Search in Sequence Databases, in Lecture Notes in Computer Science 730, Springer Verlag, 1993, 69-84.

[Das 98] Das, G., Smyth, P. et. al. Rule Discovery from Time Series. In Proc. of the 4th Int. Conf. on Knowledge Discovery and Data Mining, pp. 16-22, AAAI Press 1998.

[Doro 00] A Practical Suffix-Tree Implementation for String Searches. Dorohonceanu, B and Nevill-Manning, C. 133-140, *Dr. Dobb's Journal*, July 2000.

[Eise 98] Cluster Analysis and Display of Genome-Wide Expression Patterns. Michael B. Eisen, Paul T. Spellman, Patrick O. Brown, and David Botstein. PNAS, USA, Vol. 95, pp. 14863-14868, December 1998.

[Gusf 97] *Algorithms on Strings, Trees, and Sequence - Computer Science and Computational Biology.* Gusfield, D. Cambridge University Press, 1997.

[Holt 00] Holter, N.S., Fedoroff, N.V. et. al. Fundamental patterns underlying gene expression profiles: Simplicity from complexity. PNAS, USA July 18, 2000 Vol.97 no.15 8409-8414.

[Hui 00] A Practical Algorithm to Find Longest Common Substring in Linear Time. Lucas Chi Kwong Hui. *Computer Systems Science and Engineering*, Vol. 15, Number 2, 73-76, March 2000

[Spel 98] Spellman, P.T., Sherlock, G. et. al. Comprehensive Identification of Cell Cycle-regulated Genes of the Yeast *Saccharomyces cervisiae* by Microarray Hybridization. *Molecular Biology of the Cell*, Vol. 9, 3273-3297, December 1998.