Mining Sequential Pattern with Time-Constraint
Anita Zala, Mehul Barot

Abstract — Sequential pattern mining is an important data mining task, and different algorithms have been proposed to perform this task efficiently. The problem is to find all sequential patterns with higher or equal support to a predefined minimum support threshold in a data sequence database. Here we present a new methodology to mine a sequential pattern with time constraint. Our study shows that constraints can be effectively and efficiently pushed deep into sequential pattern mining under this new framework.

I. INTRODUCTION
Data mining is to find valid, novel, potentially useful, and ultimately understandable patterns in data. Sequential pattern mining, which extracts frequent subsequences from a sequence database, has attracted a great deal of interest during the recent surge in data mining research because it is the basis of many applications, such as customer behavior analysis, stock trend prediction, and DNA sequence analysis. The sequential mining problem was first introduced in; two sequential patterns examples are: “80% of the people who buy a television also buy a video camera within a day”, and “Every time Microsoft stock drops by 5%, then IBM stock will also drop by at least 4% within three days”. The above patterns can be used to determine the efficient use of shelf space for customer convenience, or to properly plan the next step during an economic crisis. Sequential pattern mining is also very important for analyzing bio-logical data, in which a very small alphabet (i.e., 4 for DNA sequences and 20 for protein sequences) and long patterns with a typical length of few hundreds or even thousands frequently appear. Sequence discovery can be thought of as essentially an association discovery over a temporal database. While association rules discern only intra-event patterns (item sets), sequential pattern mining discerns inter-event patterns (sequences). There are many other important tasks related to association rule mining, such as correlations, causality, episodes, multi dimensional patterns, maximal patterns, partial periodicity, and emerging patterns. Elaborate exploration of sequential pattern mining issue will be beneficial to the other research problems shown above a lot. Thus, effective and efficient sequential pattern mining is an important and interesting research problem. Efficient sequential pattern mining methodologies have been studied extensively in many related problems, including the general sequential pattern mining, constraint-based sequential pattern mining, incremental sequential pattern mining, frequent episode mining, approximate sequential pattern mining, partial periodic pattern mining, temporal pattern mining in data stream, maximal and closed sequential pattern mining. Although there are so many problems related to sequential pattern mining explored, we realize that the general sequential pattern mining algorithm development is the most basic one because all the others can benefit from the strategies it employs, i.e., Apriori heuristic and projection-based pattern growth. Hence we aim to develop an efficient general sequential pattern mining algorithm. All of these works suffer from the problems of having a large search space and the ineffectiveness in handling dense data sets, i.e., biological data. In this work, we propose new strategies to reduce the space necessary to be searched. Instead of searching the entire projected database for each item, as Prefix Span [1] does, we only search a small portion of the database by recording the last position of each item in each sequence.

II. PROBLEM FORMULATION
Sequential pattern mining is an important data mining task, and different algorithms have been proposed to perform this task efficiently. The problem is to find all sequential patterns with higher or equal support to a predefined minimum support threshold in a data sequence database.

A. Problem Statement
Let $I = \{i_1, \ldots, i_m\}$ be a set of $m$ distinct items. An event (itemset) is a non-empty subset of $I$. A sequence is a temporally ordered list of events. We denote an event as $(j_1, \ldots, j_m)$ and a sequence as $(\alpha_1 \rightarrow \ldots \rightarrow \alpha_k)$, where each $j_i$ is an item and each $\alpha_i$ is an event $(j_i \subseteq I$ and $\alpha_i \subseteq I)$. The symbol $\rightarrow$ denotes a happens-after relationship. The items that appear together in an event happen simultaneously. The length $|x|$ of a sequence $x$ is the number of items contained in the sequence. A sequence of length $k$ is called a $k$-sequence. Even if an event represents a set of items occurring simultaneously, it is convenient to assume that there exists an ordering relationship $R$ among them. Such order makes unequivocal the way in which a sequence is written, e.g., we cannot write $BA \rightarrow DBF$ since the correct way is $AB \rightarrow BDF$. This allows us to say, without ambiguity, that the sequence $A \rightarrow BD$ is a prefix of $A \rightarrow DBF \rightarrow A$, while DF $\rightarrow A$ is a suffix. A prefix/suffix of a given sequence $x$ are particular subsequence’s of $x$.

B. The Proposed Objectives
The problems or the limitations defined in the above section of this chapter are proposed to be solved by:
- To observe the effect of various existing algorithms for mining frequent sequences on various datasets.
- To propose a new method for mining the frequent sequences for sequence database i.e. for the above problem.
- To validate the new method on different datasets used in various ways.
III. PROPOSED METHOD TO FIND SEQUENTIAL PATTERN USING TIME CONSTRAINT

Assume that the DB can fit into main memory. New method first loads the DB into memory (as MDB) and scans MDB once to find all frequent items. With respect to each frequent item, New method then constructs a time set for the I-sequence items and recursively forms time-constrained sequential patterns of longer lengths. The time set is a set of (data-sequence pointer, time) pairs. For each item, only those data sequences containing that item are included in the time set. The time set is composed of times where each time is a list of triples of the form: lst (initial time, last-start time, last-end time). Mine (P, P-T) effectively locates Ps (periods); it mines type-1 patterns and type-2 patterns of prefix P. P-T is the time set for P. New method thus never searches the data sequences irrelevant to P. The Ps ensures that new method locates and counts the effective stems which can form valid patterns, rather than the whole set of items in the data sequence. When a type-1 pattern or type-2 pattern P' is formed, its time set P'-T will be constructed and Mine(P', P'-T) will be invoked recursively. By pushing time attributes deeply into the mining process, new method efficiently discovers the desired patterns.

**Input:** DB (a sequence database), minsup (minimum support), mingap (minimum gap), maxgap (maximum gap), swin (sliding time-window), duration (duration)

**Output:** the set of all time-constrained sequential patterns.

Step 1: Load DB into memory (as MDB) and scan MDB once to find all frequent items.

Step 2: for each frequent item x,

1. From the sequential pattern P = <(x)> and output P.
2. Scan MDB once to construct P-T, time set of x.
3. Call mine (P, P-T)

**Subroutine:** Mine (P, P-T)

**Parameter:** P = prefix pattern, P-T = time set

1. For each data sequence ds in the P-DB, // P-DB : sequences indicated in P-T

   (1) Use the corresponding time to collect the type-1 Ps satisfying:

   \[ \text{let } i, \text{ mingap} \leq \text{period} \leq \text{min} \{\text{lst}, \text{ maxgap}, \text{ it}, \text{ duration}\}, \ i, 1 \leq i \leq k \]

   (2) For each item in the Ps of type-1 pattern, add one to its support count.

   (3) use the corresponding time to collect the type-2 Ps satisfying :

   \[ \text{let} \ i, \ \text{swin} \leq \text{period} \leq \text{min} \{\text{lst}, \text{ swin, it}, \text{ duration}\}, \ i, 1 \leq i \leq k \]

   (4) for each item in the Ps of type-2 pattern, add one to its support count

2. for each item x’ found in the Ps of type-1 pattern and its support is greater than or equal to minsup,

   (1) form of type-1 pattern P’ by extending stem x’ and output P’.

   (2) scan the Ps of each ds in P-DB to construct P’-T, time set of x’.

   (3) call Mine(P’, P’-T);

3. for each item x’ found in the Ps of type-2 pattern and its support is greater than or equal to minsup,

   (1) form of type-2 pattern P’ by appending stem x’ and output P’.

   (2) scan the Ps of each ds in P-DB to construct P’-T, time set of x’.

   (3) call Mine(P’, P’-T);

IV. EXAMPLE

Given a DB in Table 1, minsup = 50%, mingap = 3, maxgap = 15, swin = 2 and duration = 25, new method mines sequential patterns by the following steps:

**Table 1: Sequence Database (DB)**

<table>
<thead>
<tr>
<th>Sid</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>&lt;(c)&gt;(a,f)&gt;(b)&gt;(a)&gt;(d)&gt;</td>
</tr>
<tr>
<td>C2</td>
<td>&lt;(a,c)&gt;(b)&gt;(c)&gt;(d)&gt;(e)&gt;</td>
</tr>
<tr>
<td>C3</td>
<td>&lt;(b)&gt;(d,g)&gt;(e)&gt;(g)&gt;</td>
</tr>
<tr>
<td>C4</td>
<td>&lt;(a)&gt;(d)&gt;(c,d)&gt;(e)&gt;</td>
</tr>
</tbody>
</table>

Step 1: Load DB into memory and find all frequent items. New method first reads the DB into memory and scans the in-memory DB (abbreviated as MDB) once to find frequent items. <(a)>:3, <(b)>:3, <(c)>:3, <(d)>:3, <(e)>:3 are found, <(a)>:3 shows its support count of 3. For each frequent item, new method applies the following steps.
Step 2: Construct the time set of the frequent item. New method scans MDB once to construct the time set; that set contains the times and the sequence pointers at which the frequent items appear.

Take \(<(a)>\) for example, new method scans MDB and constructs the time set \(<(a)>-T\), as shown in Table 2. Three pointers appear in \(<(a)>-T\) since \(<(a)>\) appears in C1, C2, and C4. For C1 \(<(c)>(a, f)_{18}(b)_{31}(a)_{45}(f)>\), element \((a)\) occurs at time 5 and 31 so the time is marked as [5:5:5, 31:31:31]. The times for C2 and C4 are processed similarly.

Step 3: Find stems in Ps from the time set and grow (discover) patterns. For each sequence containing P, new method uses the time to find the Ps of P. For type \(2\) patterns, new method uses the time to find the Ps of P. New method labels sequence pointers containing P'.

The \(<(a) (b)>\)-T is constructed and sub-routine Mine is called recursively. Table 3 shows \(<(a) (b)>-T\). With regard to the type-1 Ps of \(<(a) (b)>\), which are [21:33] for C1 and [13:25] for C2, no more stems can be found. No pattern of prefix \(<(a)(b)>\) can be formed.

Table 3: \(<(A) (B)>-T\)

<table>
<thead>
<tr>
<th>Sids</th>
<th>Sequences</th>
<th>Time set</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>(&lt;(c)&gt;(a, f)<em>{18}(b)</em>{31}(a)_{45}(f)&gt;)</td>
<td>[5:5:5, 31:31:31]</td>
</tr>
<tr>
<td>C2</td>
<td>(&lt;(a,c)<em>{10}(b)</em>{15}(c)<em>{18}(a)</em>{21}(d,c)&gt;)</td>
<td>[6:6:6, 18:18:18]</td>
</tr>
<tr>
<td>C3</td>
<td>(&lt;(b)<em>{20}(b,g)</em>{26}(e)<em>{18}(d,g)</em>{35}(g)&gt;)</td>
<td>----</td>
</tr>
<tr>
<td>C4</td>
<td>(&lt;(a)<em>{10}(d)</em>{21}(c,d)_{20}(e)&gt;)</td>
<td>[5:5:5]</td>
</tr>
</tbody>
</table>

Using \(<(a)>-T\), new method finds all the stems quickly and grows type-1 patterns and type-2 patterns. Sub-routine Mine computes the supports of potential stems in this step. With respect to prefix \(<(a)>\), the type-1 period for C1 is [8:20, 34:46] from [let1 + mingap:let1 + maxgap, let2 + mingap:let2 + maxgap], where let1 = let1 = 5 and let2 = let2 = 31. The Ps of C2 [9:21, 21:33] and C4 [8:20] are obtained similarly. Likewise, the type-2 period for C1 is [3:7, 29:33] from [let1 + mingap:let1 - swin, let2 + mingap:let2 - swin]; C2 has [4:8, 16:20], and C4 has [3:7]. Now, items b and d, whose supports pass the threshold, become new type-1 stems, and item c (whose support passes the threshold) becomes a new type-2 stem. Thus, pattern \(<(a) (b)>\) is output. New method uses step 4 to construct the time set of \(<(a) (b)>\) and grow the pattern in sub-routine Mine. After processing \(<(a) (b)>\), \(<(a) (d)>\) and \(<(a, c)>\) are processed in turn using the same process.

Step 4: Recursively construct the time set of the sequential pattern and discover all sequential patterns. For each sequential pattern \(P^*\) with prefix \(P\) and stem \(x\), new method scans each data sequence containing \(P\) and records all the initial times, last-start times and last-end times of \(P^*\) within the periods of \(P\). New method labels sequence pointers containing \(P^*\).
Table 5: Time-Constrained Sequential Pattern

<table>
<thead>
<tr>
<th>Time-constrained sequential pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;(a)&gt;:3, &lt;(a) (b)&gt;:2, &lt;(a) (d)&gt;:2, &lt;(a, c)&gt;:2,</td>
</tr>
<tr>
<td>&lt;(a, c) (b)&gt;:2, &lt;(b)&gt;:3, &lt;(b) (a)&gt;:2, &lt;(b) (d)&gt;:2,</td>
</tr>
<tr>
<td>&lt;(b) (e)&gt;:2, &lt;(b) (e) (d)&gt;:2, &lt;(c)&gt;:3, &lt;(c) (b)&gt;:2,</td>
</tr>
<tr>
<td>&lt;(c) (e)&gt;:2, &lt;(c,d)&gt;:2, &lt;(d)&gt;:3, &lt;(e)&gt;:3, &lt;(e) (d)&gt;:2</td>
</tr>
</tbody>
</table>

V. CONCLUSION

In this paper we have develop an efficient and scalable sequential pattern based on time-constraint. It constructs a time set for the 1-sequence items and recursively forms time-constrained sequential patterns of longer lengths. By pushing time attributes deeply into the mining process, new method efficiently discovers the desired patterns.

REFERENCES


