Timed Sequential Pattern Mining Based on Confidence in Accumulated Intervals

Chichang Jou, Huan-Jyh Shyur, Chih-Yu Yen
Department of Information Management, Tamkang University
151, Yingzhuan Rd.
Tamsui, New Taipei City 25137, Taiwan
cjou@mail.tku.edu.tw, {shyur, arlu.yen}@mail.im.tku.edu.tw

***Abstract*- Many applications of sequential patterns require a guarantee of a particular event happening within a period of time. We propose CAI-PrefixSpan, a new data mining algorithm to obtain confident timed sequential patterns from sequential databases. Based on PrefixSpan, it takes advantage of the pattern-growth approach. After a particular event sequence, it would first calculate the confidence level regarding the eventual occurrence of a particular event. For those pass the minimal confidence requirement, it then computes the minimal time interval that satisfies the support requirement. It then generates corresponding projected databases, and applies itself recursively on the projected databases. With the timing information, it obtains fewer but more confident sequential patterns. CAI-PrefixSpan is implemented along with PrefixSpan. They are compared in terms of numbers of patterns obtained and execution efficiency. Our effectiveness and performance study shows that CAI-PrefixSpan is a valuable and efficient approach in obtaining timed sequential patterns.**

I. Introduction

With the universal deployment of internet and computer technologies, the amount of accumulated electronic data has been growing exponentially. Sequential pattern mining is one of the successful data mining endeavors that extracts implicit information inside these data. It obtains frequent sequential patterns of items satisfying the condition that the number of their occurrences, called *support*, in the item sequence database is greater than or equal to a given threshold, called *minimum support*. The obtained frequent patterns could be applied to analysis and decision making in applications like time-series stock trend, medical diagnosis, web page traversal, customer purchasing behavior, content signature of network applications, etc.

The existing sequential pattern mining algorithms can be separated into two categories: Apriori-like (candidate-generation-and-test) approaches ([1],[2],[10]) and pattern-growth approaches ([6],[8]). The PrefixSpan algorithm [8] divides the database into smaller projected databases and solves them recursively. Since no candidate sequence needs to be generated, the database need not be scanned multiple times, thus making it faster than Apriori-like algorithms.

A timed sequential pattern could provide more valuable information than a conventional sequential pattern. The issue of mining sequential patterns with time constraints was first addressed in [10]. Three time constraints, minimum-gap, maximum-gap and sliding time-window, were specified to enhance the semantics of sequence discovery, and to make the obtained patterns more actionable. Since then, several studies about mining sequential patterns with miscellaneous time constraints have been proposed.

In addition to the minimum support constraint, many applications of sequential patterns need to be confident in terms of the percentage of a specific event occurring within a time interval. For example, after a customer's purchasing of products A and B, a retailer would like to know the percentage that he/she would return to buy product C within a week. The guarantee is often more important than the timing. Most users are willing to extend the time interval to meet the minimal confidence requirement. We thus propose to check the percentage that a specific event happens eventually first. After that is confirmed, then we calculate the minimum time required to satisfy the minimum support requirement. We take advantage of the fact that time intervals have an intrinsic nature: suppose time point $t_1 < t_2$. Then for events guaranteed to happen within t_1, they are also guaranteed to happen within t_2. Previous tackles over timed sequential patterns failed to capture this containment nature of accumulated time intervals.

Applying the pattern growth approach, we propose CAI-PrefixSpan to handle the timed sequential pattern mining. A pattern is extended by first satisfying a minimum confidence requirement that certain event would happen eventually. Then, we compute the minimal time interval that satisfies the minimum support requirement. A projected database would then be obtained for the sequence in the extended timed sequential pattern.

CAI-PrefixSpan is implemented along with PrefixSpan. They are compared in terms of numbers of patterns obtained and execution efficiency. Our effectiveness and performance study shows that CAI-PrefixSpan is a valuable and efficient approach in obtaining timed sequential patterns.

The rest of this paper is organized as follows: Section II reviews literature on sequential pattern mining and timed sequential pattern mining. Section III describes CAI-PrefixSpan. Section IV presents the experimental results. Finally, Section V concludes and discusses future research.

II. Related Work

Agrawal and Srikant [2] extended the frequent itemset mining algorithm [1] for non-serial transactions to sequential pattern mining for serial transactions. With their approach, candidate frequent sequential patterns can be obtained by joining shorter frequent sequential patterns. Srikant and Agrawal [10] then proposed the GSP (generalizations and performance improvements) algorithm, which uses a breadth-first search and bottom up method to obtain the frequent subsequences. It also considers mining sequential patterns with timing constraints regarding the minimal time gap, maximal time gap, and sliding window size.

Chen et al., [3] developed two algorithms (I-Apriori and I-PrefixSpan) for mining time-interval sequential patterns. They assumed the time interval has already been partitioned into a set of fixed time intervals. Fiot et al. [4] utilized fuzzy set to extend the time constraint to soft time constraints. They defined temporal accuracy of a sequential pattern. To handle these constraints while, they designed a data mining algorithm based on sequence graphs. Masseglia et al. [7] considered handling time constraints in the earlier stage of the data mining process to provide better performance.

Guyet and Quiniou [5] handled the problem of quantitative temporal pattern extraction from temporal interval sequences where events are qualified by a type and a numerical date and duration. Yang et al. [11] adopted an efficient encoding strategy to speed up the efficiency of processing period segments in an event sequence, and combined with the projection method to quickly find the partial periodic patterns in the recursive process. Shyur et al. [9] proposed to discover frequent sequential patterns with probability of inter arrival time of consecutive items. They imposed minimum time-probability constraint on sequential patterns, so that fewer but more reliable patterns will be obtained.

III. CAI-PrefixSpan Algorithm

A. *Introduction*

Chen et al. [3] proposed I-PrefixSpan to partition time intervals into several equal-length sub-intervals. Concept of the partitioned time intervals could be illustrated in Figure 1.

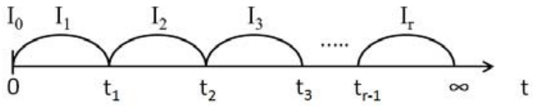

Fig. 1 Partitioned time intervals

In Figure 1, I_0 represents the instantaneous interval $0 \leq t \leq 0$. A pattern < e_1, I_0, e_2> means events e_1 and e_2 happen at the same time.

Let e_1, e_2, e_3 denote events. And let I_1 and I_2 denote two time intervals that do not overlap. Their frequent sequential patterns are of the form <e_1, I_1, e_2, I_2, e_3>, which means that after the occurrence of event e_1, event e_2 would occur within time interval I_1, and then event e_3 would occur within time interval I_2. A timed sequence is said to support a timed sequential pattern if the events in the sequential pattern also appear in the timed sequence with the same order and the time differences between adjacent events are within each corresponding interval. Based on checking the support counts of patterns occurring in each interval, they obtain the frequent patterns by extending the patterns step by step with an event and an interval.

Their separate counting for partitioned (or non-overlapping) intervals makes it possible that the same transaction be counted as supports several times with different intervals, which makes the meaning of support counting unclear. On the other hand, due to separate support counting of disjoint time intervals, frequent patterns obtained in PrefixSpan might not pass the minimum support requirement. Thus, they would not be collected as frequent patterns in I-PrefixSpan. Additionally, their patterns could not capture the following intrinsic nature of "*containment*" relationship among time intervals: Suppose time point $t_1 < t_2$. An event happens within t_1 implies that it also happens within t_2.

The length of the time intervals would affect the discovered results in I-PrefixSpan. If the length is set too long, then the timing information in the obtained patterns do not convey enough information. If the length is set too short, then most patterns would not pass the minimum threshold.

In many applications, the existence of more than one in-between time intervals in I-PrefixSpan is too detailed to make it useful. For example, in diagnosing a disease, after appearances of several symptoms, a doctor would rather be confident about symptoms would appear afterwards within certain time interval. Thus, we propose that in a timed sequential pattern, the prefix part of the event sequence does not have to include the timing information. Only the interval between the prefix and the last event needs to be specified.

In our model, the user could specify the cut-off-time that he/she would like to collect timing information. Beyond the specified cut-off-time, they are only interested in whether the event would happen eventually. A timed pattern is of the form <α, I, e>, where α is an event sequence, I is an interval of the form $0 \leq t \leq t_i$ for some time point t_i, and e is the event happening after the occurrence of α. We call such intervals *accumulated intervals*. They satisfy the intrinsic "*containment*" relationship. Concept of the accumulated intervals is illustrated in Figure 2.

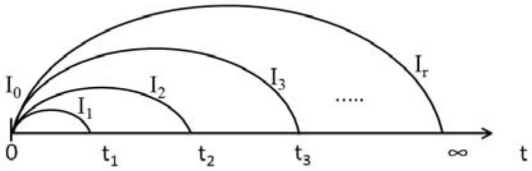

Fig. 2 Accumulated time intervals

Our model could handle the cut-off-time cases by setting the next-to-last interval as $0 < t \leq t_c$, for some cut-off-time t_c, and setting the last interval as $0 < t < \infty$. Granularity of the time intervals could be adjusted based on the required accuracy level, and it would not split the support counting, which happened in I-PrefixSpan.

B. *Model*

We use purchasing transactions of customers in a retail store as an example to explain the CAI-PrefixSpan algorithm.

Given a set of items **A**, a timed sequence is represented as $<(a_1, t_1), (a_2, t_2), \ldots, (a_n, t_n)>$, where for all $1 \leq j \leq n$, $t_{j-1} \leq t_j$, a_j is an item in **A**, and t_j is the time point at which purchasing of a_j occurs. Since a customer might purchase more than one item in a transaction, we model this kind of transaction by ordering the items alphabetically and by assigning the same time points for them.

The accumulated intervals $I_0, I_1, I_2, \ldots, I_r$ for a sequence of time points $t_0, t_1, \ldots, t_{r-1}, t_r$ are defined as follows:

- $t_0 = 0$, $t_r = \infty$, and for all $1 \leq i \leq r\text{-}1$, $t_i < t_{i+1}$.
- I_0 is for purchasing events happening in the same transaction, that is $t = 0$.
- I_j is used to denote that the time difference t between two items satisfies $0 < t \leq t_j$.

Time point t_{r-1} is the cut-off-time. In the following, we will denote I_r, the only time interval that includes all the time after the cut-off-time, as I_∞. The accumulated intervals satisfy the following containment property:

- for all $1 \leq i < r\text{-}1$, $I_i \subset I_{i+1}$.
- for all $1 \leq i < r$, a discovered pattern $<\alpha, I_i, b>$ means after the event sequence α, it is quite possible to purchase item b within interval I_i.

In our model, when we have the extreme pattern $<\alpha, I_\infty, b>$ but not $<\alpha, I_{r-1}, b>$, it means that starting from the last purchasing of the event sequence α, after at least I_{r-1}, purchasing of item b would happen with a guarantee above the minimum confidence requirement.

Definition 1: Let $\boldsymbol{A} = \{ a_1, a_2, \ldots, a_m \}$ be the set of all items and $\boldsymbol{TI} = \{ I_0, \ldots, I_{r-1}, I_\infty \}$ be the set of all accumulated time intervals. A sequence $X = <x_1, x_2, \ldots, x_{k-1}, I, x_k>$ is a *sequential pattern* if for all $1 \leq i \leq k$, $x_i \in \boldsymbol{A}$, and $I \in \boldsymbol{TI}$. The number of events in X, that is k, is called the *length* of X.

Definition 2: A sequential pattern X is said to be *contained in a sequence* $s=<(a_1, t_1), (a_2, t_2), \ldots, (a_n, t_n)>$,, denoted as $X \subset s$, if exist integers $1 \leq j_1 < j_2 \ldots < j_k \leq n$ such that:

1. $a_1 = x_{j_1}, \ldots, a_k = x_{j_k}$
2. time difference $t_{j_k} - t_{j_{k-1}}$ is within the interval I.

Definition 3: For a timed sequence database $\boldsymbol{S}$, *support count* of a sequential pattern X *with respect to* $\boldsymbol{S}$ is defined as the number of sequences in which X is contained:

$$support(X, \boldsymbol{S}) = \left| \{ s_{id} \mid X \subset s_{id}, s_{id} \in S \} \right|$$

A pattern X is called a *frequent timed sequential pattern* in a database $\boldsymbol{S}$ if $support(X, \boldsymbol{S})$ is greater than the minimum support.

Definition 4: *Confidence* of an event sequence $\alpha = <x_1, x_2, \ldots, x_{k-1}, x_k>$ is defined as follows:

$$confidence(\alpha) = \frac{\left| \{ s_{id} \mid <x_1, x_2, \ldots, x_{k-1}, I_\infty, x_k> \subset s_{id} \} \right|}{\left| \{ s_{id} \mid <x_1, x_2, \ldots, x_{k-2}, I_\infty, x_{k-1}> \subset s_{id} \} \right|}$$

Confidence of α captures the percentage that the event x_k happens after the occurrence of event sequence $<x_1, \ldots, x_{k-1}>$. From the association rule mining, the confidence level is very important for applications.

Definition 5: *Confidence of a sequential pattern* $X = <x_1, x_2, \ldots, x_{k-1}, I, x_k>$ is defined as the confidence of the event sequence $\alpha = <x_1, x_2, \ldots, x_{k-1}, x_k>$.

Note that confidence of a sequential pattern is independent of its time interval I.

In the extending phase of CAI-PrefixSpan algorithm, suppose we have found a pattern of the form $<x_1, x_2 \ldots, I, x_{k-1}>$. We will first find the set of frequent items in the projected database $S_{<x_1, x_2, \ldots, x_{k-1}>}$. Then for each frequent item b, we first calculate the confidence level of the extended sequence $<x_1, \ldots, x_{k-1}, b>$. They represent the percentage that item b are guaranteed to *eventually* happen after the event sequence $<x_1, \ldots, x_{k-1}>$. For those items passing the requirement, we then calculate the minimal time interval that satisfies the minimum support requirement.

With the concept of accumulated intervals, the support counting mechanism would be as follows: with the time interval as the row and the sequence id as the column, we check the containment of a sequential pattern with respect to each sequence for each accumulated interval. The entry would be 1 if an event occurs within that time interval. Note that if the entry for a time interval I_i is 1, then the entry for all intervals containing I_i would be 1. To explain the support counting mechanism, we use the sequential database in Table I.

TABLE I
Example Database for Minimal Time Interval Calculation

Sequence id	**Timed Sequence**
10	<(a,3)(d,8)>
20	<(a,1)(d,2)>
30	<(a,4)(d,11)>

Suppose I_1: $0 < t \leq 2$, I_2: $0 < t \leq 4$, I_3: $0 < t \leq 6$, I_4: $0 < t < \infty$, and the minimum support is 2. Starting with I_0, from small to large accumulated intervals, we find the minimal accumulated interval satisfying the minimum support requirement. The calculation of minimal time interval satisfying minimal support of a sequence $<a, d>$ is displayed in Table II.

In Table II, I_3 is the shortest accumulated interval satisfying the minimum support requirement. We thus obtain the sequential pattern $<a, I_3, d>$ for the sequence $<a, d>$.

For data mining systems with a low minimal support threshold, the inclusion of the confidence constraints is not only to eliminate less confident patterns, but also to keep rare but important patterns.

TABLE II
Example Calculation of Minimal Time Interval

id interval	10	20	30	Total support
I_0	0	0	0	0
I_1	0	1	0	1
I_2	0	1	0	1
I_3	1	1	0	2
I_4	1	1	1	3
1: the sequence is in the accumulated interval 0: the sequence is not in the accumulated interval				

C. The CAI-PrefixSpan Algorithm

We extend PrefixSpan with the confidence constraints in accumulated intervals to obtain the CAI-PrefixSpan algorithm. With the pattern growth approach, CAI-PrefixSpan would extend currently obtained patterns, one item at a time, with another frequent item. It would then impose the confidence constraint to ensure that the percentage of the eventual support of the newly generated patterns and that of the current patterns is greater than or equal to the minimum confidence threshold. Our support count would apply only to those patterns passing the constraints.

In CAI-PrefixSpan, we would first scan the whole timed sequential database to record number of occurrences of each item. After eliminating those items with occurrence below the minimum support, we obtain level-one sequential patterns L_1. Then we build the projected databases for each item in L_1, and record the happening time of the item. For each event sequence, we would only record the one happening at the earliest time. That would prevent redundant support counting for a timed sequence. Then, in the projected databases, we would extend them with a frequent item, and calculate the confidence level of the resulting L_2 sequence. If they pass the minimum confidence constraints, then we would perform their support counting mechanism for accumulated intervals. After that, projected databases for the extended sequences would be generated. The above procedures would be applied recursively to the projected databases until no more projected databases could be generated.

Definition 6: Given an event sequence $s=<(a_1, t_1), (a_2, t_2), \ldots, (a_n, t_n)>$ and a timed sequential pattern $X=<(b_1, b_2, \ldots, b_{k-1},, I, b_k>$ *(for some* $k \leq n$*)*. Let $i_1 < i_2 < \ldots < i_s$ be the indexes of the elements in s that matches the elements of X. A subsequence $s' = <(a'_1, t'_1), (a'_2, t'_2), \ldots, (a'_p, t'_p)>$ of s, where $p = k + n - i_s$, is called a *projection of s with respect to X* if and only if the last $n\text{-}i_s$ elements of s′ are the same as the last $n\text{-}i_s$ elements of s.

Note that there might exist more than one projection sequences for a pair of event sequence and a timed sequential pattern. We would choose the one with the earliest occurrence time, which will contain all the relevant information.

Definition 7: For a timed sequential pattern X, the set of projections of all sequences in the original database $\boldsymbol{S}$ with respect to X is called *the projected database of* $\boldsymbol{S}$ *with respect to* X, and is denoted as S_X.

Figure 3 and 4 display pseudo codes of the CAI-PrefixSpan algorithm. Given timed sequential database $\boldsymbol{S}$, we would call *CAI-PrefixSpan(< >,**S**)* to obtain the set of all timed sequential patterns.

```
CAI-PrefixSpan($X$, $S_X$)
Input: a sequential pattern $X$, projected timed-sequential database $S_X$.
// note that if $X$ = "< >", then $S_X = S$
Parameters: minimum support min_sup, minimum confidence min_conf,  set of accumulated intervals $\boldsymbol{TI}$
Output: All the frequent timed sequential patterns satisfying both the min_sup and min_conf conditions and with prefix equal to the event sequence in $X$
Steps:
1. Scan $\boldsymbol{S_X}$ once to obtain all frequent items in $\boldsymbol{S_X}$
2. for each frequent item $b$:
   if $X$ = "< >" then let $X' = <b>$.
   else {
      let $\alpha$ be the event sequence in $X$
      append $b$ to $\alpha$ to generate an extended sequence $\alpha'$
      if $confidence(\alpha') \geq min\_conf$, then {
         $I_{\alpha'} = Get\text{-}TimeInterval(\alpha')$  //Figure 4
         let $X' = <\alpha, I_{\alpha'}, b>$
     }
      output $X'$
   }
   Build the projected database $\boldsymbol{S_{X'}}$.
   If $\boldsymbol{S_{X'}}$ is not $\emptyset$, call CAI-PrefixSpan($X'$, $\boldsymbol{S_{X'}}$)
```

Fig. 3 Pseudo codes of CAI-PrefixSpan

```
Function Get-TimeInterval($\alpha$)
Input: an event sequence $\alpha = <x_1, x_2, \ldots, x_{k-1}, x_k>$
Parameters: minimum support min_sup, set of accumulated intervals $\boldsymbol{TI}$
Output: the shortest accumulated interval I such that $support\ (<x_1, x_2, \ldots, x_{k-1}, I, x_k>) \geq min\_sup$
Steps:
For all $I \in \boldsymbol{TI}$, from small to large $I$
   If $support\ (<x_1, x_2, \ldots, x_{k-1}, I, x_k>) \geq min\_sup$  then
      return $I$
EndFor
```

Fig. 4 Pseudo code of Get-TimeInterval

We illustrate the CAI-PrefixSpan algorithm with the example database in Table III. Parameters are min_sup = 2, min_conf = 50%, and accumulated intervals I_1*: 0 < t ≤ 2,* I_2*: 0 < t ≤ 4,* I_3*: 0 < t ≤ 6,* I_4*:0 < t < ∞.*

TABLE III
Example Database for CAI-SprefixSpan

Sequence id	Timed Sequence
10	<(b,1) , (a,3) , (e,4) , (b,6) , (e,6) >
20	<(a,3), (b,4) , (e,4) , (b,7) , (e,9) >
30	<(a,5), (c,8) , (a,10) , (e,10) , (a,11) , (c,14)>
40	<(b,7) , (f,9) , (a,15) , (b,16) , (e17) , (f,17)>
50	<(a,8) , (d,10) , (f,10) , (a,17) , (f,18)>

Step 1: Obtain the set of all frequent items L_1. The frequent items and their support counts are L_1 = { *a:5, b:3, e:4, f:2* }.

Step 2: Build projected databases for each item in L_1. Table IV shows the obtained projected databases.

TABLE IV
Projected Databases for Level 1 Sequential Patterns

Prefix	ID	Starting time	Projected database
<a>	10	3	< (e,4) , (b,6) , (e,6) >
	20	3	< (b,4) , (e,4) , (b,7) , (e,9) >
	30	5	< (c,8) , (a,10) , (e,10) , (a,11) , (c,14)>
	40	15	< (b,16) , (e,17) , (f,17)>
	50	8	< (d,10) , (f,10) , (a,17) , (f,18)>
<b>	10	1	< (a,3) , (e,4) , (b,6) , (e,6) >
	20	4	< (e,4) , (b,7) , (e,9) >
	40	7	< (f9) , (a,15) , (b,16) , (e17) , (f,17)>
<e>	10	4	< (b,6) , (e,6) >
	20	4	< (b,7) , (e,9) >
	30	10	< (a,11) , (c,14)>
	40	17	< (f,17)>
<f>	40	9	< (a,15) , (b,16) , (e17) , (f,17)>
	50	10	< (a,17) , (f,18)>

Step 3: Recursively call CAI-Prefix(X′, $S_{X'}$) for each X′ with length 1. In the following, we use <*a*> as an example.

- **Step 3.1** the set of all frequent items in $S_{<a>}$ are: { *a:2, b :3, e:4, f:2* }.
- **Step 3.2** Confidence constraint filtering: Since the confidence of <*a,a*> and <*a,f*> are both *2/5*, which is less than min_conf. Therefore, only items *b* and *e* pass the 50% min_conf condition. Therefore, only event sequences <*a,b*> and <*a,e*> will be considered next.

Step 4: Get-TimeInterval(α′): Use <*a,b*> as an example. Table V shows the support counting for each accumulated interval.

TABLE V
Example Calculation of Minimal Time Interval for <*a,b*>

ID Interval	10	20	40	Total support
I_0	0	0	0	0
I_1	0	1	1	2
I_2	1	1	1	3
I_3	1	1	1	3
I_4	1	1	1	3

From Table V, we could see that I_1 is the shortest accumulated time interval within which <*a, b*> satisfies the min_sup condition. Similarly, I_1 is also the shortest accumulated time interval within which <*a, e*> satisfies the min_sup condition. Thus, we have the following level-2 patterns with prefix <*a*>: $L_{2,<a>}$ = { <*a*,I_1,*b*>, <*a*,I_1,*e*> }.

Step 5: Recursively call CAI-Prefix(*X'*, $S_{X'}$) for each *X'* in L_2. Using the same methods, we would obtain the set of all frequent patterns with prefix <*a*> : { <*a*, I_1, *b*> , <*a*, $I_{1.}$, *e*> , <*a, b*, I_0, *e*>, <*a, e*, I_0, *b*> <*a, b*, I_3, *e*> , <*a, e*, I_2, *b*> , <*a, e*, $I_{3.}$, *e*> }.

C. *Comparisons of PrefixSpan, I-PrefixSpan, and CAI-PrefixSpan*

Table VI shows the results obtained from the PrefixSpan, I-PrefixSpan, and CAI-PrefixSpan algorithms. With the partitioned time interval, the number of frequent patterns obtained from I-PrefixSpan is less than that from PrefixSpan. On the other hand, CAI-PrefixSpan only eliminate <*a,a*> and <*a,f*>, since they do not satisfy the min_conf constraint. Additionally, I-PrefixSpan would obtain patterns like <*a*, I_1, *e*> and <*a*, I_3, *e*>. On the other hand, with the concept of accumulated interval, CAI-PrefixSpan would only have <*a*, I_1, *e*>, which implies for all *I* containing I_1, <*a, I, e*> would hold.

We could see from the comparison that unlike I-PrefixSpan, CAI-PrefixSpan would not lose results obtained in PrefixSpan, except those that do not pass the min confidence condition.

The pattern <*a,(be)*> obtained in PrefixSpan are now captured by the two timed sequential patterns in CAI-PrefixSpan: <*a, b*, I_0, *e*> and <*a, e*, I_0, *b*>. The pattern <*a,b,e*> in PrefixSpan is not obtained in CAI-PrefixSpan, since <*a, b*, I_0, *e*> would cover <*a, b, I, e*> for all *I*. For cases that the users would like to distinguish between <*a,(be)*> and <*a, b, e*> in the timed sequential patterns, studies about handling the issues regarding combination of the “simultaneity” and “containment” relationships are under way.

TABLE VI
Comparison of Obtained Sequential Patterns

Algorithm	**Level of sequential patterns**			**Total number of sequential patterns**
	1	**2**	**3**	
PrefixSpan	*<a>* *<b>* *<e>* *<f>*	*<a,a>* *<a,b>* *<a,e>* *<a,f>* *<b,a>* *<b,b>* *<b,e>* *<(be)>* *<e,b>* *<f,a>* *<f,f>*	*<a,(be)>* *<a,b,e>* *<a,e,b>* *<a,e,e>* *<b,a,b>* *<b,a,e>* *<b,a,b>* *<b,b,e>* *<f,a,f>*	24
I-PrefixSpan	*<a>* *<b>* *<e>* *<f>*	<a,I_1, b> <a,I_1, e> <a,I_1, f> <a,I_2,b> <a,I_3,e> <b,I_0, e> <b,I_1, e> <b,I_3,e> <f,I_4, f>	*None*	13
CAI-PrefixSpan	*<a>* *<b>* *<e>* *<f>*	<a,I_1,b> <a,I_1,e> <b,I_4,a> <b,I_3,b> <b,I_0,e> <e,I_0,b)> <e,I_3,e> <f,I_4,a> <f,I_4, f>	<a,b I_0,e> <a,e I_0,b> <a,e,I_3,e> <b,a,I_1,e> <b,a,I_2,b> <b,b,I_1,e> <f,a,I_1, f>	20

IV. EXPERIMENTS

To evaluate the performance and efficiency of CAI-PrefixSpan, we implement CAI-PrefixSpan, along with PrefixSpan, in C++ under Windows XP Professional operating systems, with AM2 3800+ 2.01GHz CPU and 1G DRAM memory. The sequential databases are generated with the open source codes from the Quest project in IBM Almaden Lab. The starting time and the duration between two transactions in a sequence are produced with two Poisson distributions.

We have three experiments with the following purposes: (1) Compare the performance and obtained patterns under different minimum support between CAI-PrefixSpan and PrefixSpan. (2) Examine the influence of database size to CAI-PrefixSpan and PrefixSpan to check the scalability of CAI-PrefixSpan. (3) Examine the influence of confidence to CAI-PrefixSpan with respect to performance and obtained patterns. The parameters we use in these experiments are in Table VII.

TABLE VII
Parameters of the Experiment Databases

Parameter	**Explanation**
D	Number of sequences
N	Number of items
C	Average length of a sequence
T	Average number of items in a transaction
S	Average number of items in a sequential pattern
I	Average number of items in an association rule

Experiment 1: With D=30000, N=1000, C=10, T=5, S=4, I=2.5, minimum confidence=20%, we have the performance and patterns for different minimum support displayed in Figure 5 and Figure 6.

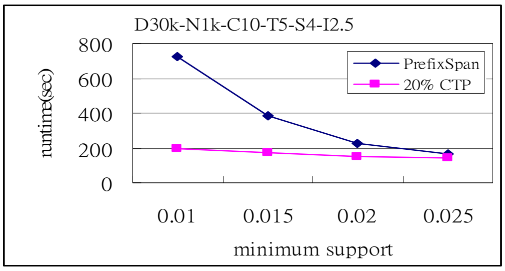

Fig. 5 Comparison of Execution Time by Minimum Support

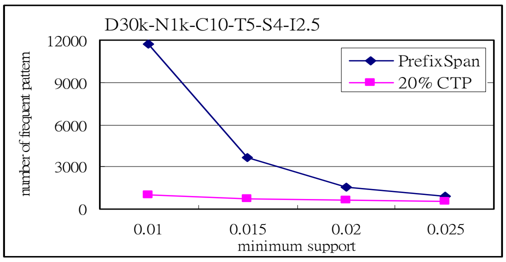

Fig. 6 Comparison of Number of Patterns by Minimum Support

From Figure 5, we could find that execution time in CAI-PrefixSpan is always smaller than that of PrefixSpan, especially when the minimum support is small. The reason is that CAI-PrefixSpan imposes the confidence constraints to filter out un-reliable patterns. Especially, the filtering in L_2 could reduce the number of projected databases dramatically. The smaller the minimum support, the more patterns obtained in L_2 in PrefixSpan. The confidence constraints would reduce search space, and thus would reduce execution time.

Experiment 2: With N=1000, C=10, T=5, S=8, I=2.5, minimum support = 0.01, minimum confidence=20%, we have the performance and number of obtained patterns for database size 10000 to 50000 displayed in Figure 7 and Figure 8.

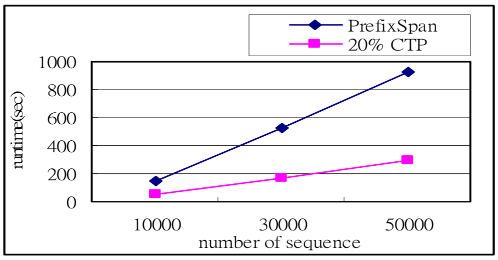

Fig. 7 Comparison of Number of Patterns by Database Size

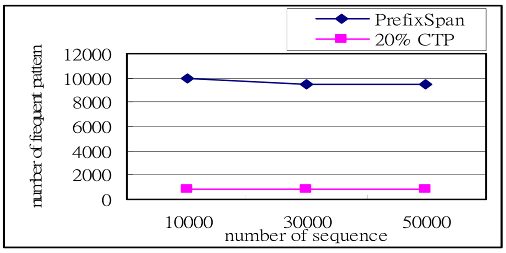

Fig. 8 Comparison of Number of Patterns by Database Size

From Figure 7, the slope for PrefixSpan is steeper than that for CAI-PrefixSpan. When the database size increases, the execution time of PrefixSpan would be affected severely. From Figure 8, in both PrefixSpan and CAI-PrefixSpan, the numbers of sequential patterns are about the same for different database sizes. However, for each database size, the difference in number of sequential patterns obtained by PrefixSpan and CAI-PrefixSpan is huge.

Table VIII shows detailed number of sequential patterns with different lengths for database size = 10000, and minimum confidence=20%.

TABLE VIII
Comparisons of Number of Patterns in Each Level

Algorithm	**Level of Sequential patterns**				**Total number of sequential patterns**
	1	**2**	**3**	**4**	
PrefixSpan	766	9201	6	1	9974
CAI-PrefixSpan	766	109	1	None	876

From Table VIII, we found that in PrefixSpan, there are 9201 sequential patterns with length 2, while the minimum confidence of 20% in CAI-PrefixSpan would keep only 109 sequential patterns. In this experiment, the filtering is caused from the parameters that the average number of items in a sequential pattern is 8, while Average number of items in an association rule is 2.5.

Experiment 3: With D=30k, N=1k, C=10, S=4, minimum support = 0.005, we have the performance and number of obtained patterns for different minimum confidence. Figures 9 and 10 shows the results for combinations of average number of items in a transaction (T) and average number of items in an association rule (I).

From Figure 10, we found that when the minimum confidence level is increased from 0.2 to 0.4, the drop in number of patterns is more severe than the case from 0.4 to 0.6. However, the execution time reduction Figure 9 is not so obvious. In other words, requirements of minimum support and minimum confidence would affect each other. Users might have to try different combinations to find suitable parameters.

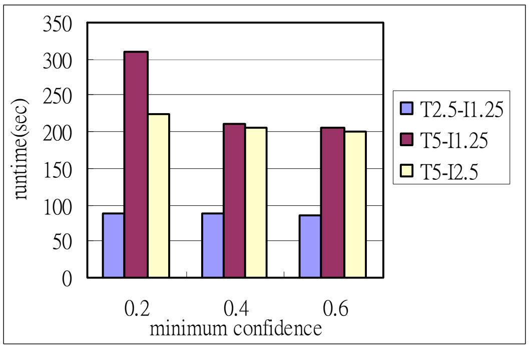

Fig. 9 Comparison of Execution Time by Average Number of Items in a Transaction and Number of Items in an Association Rule

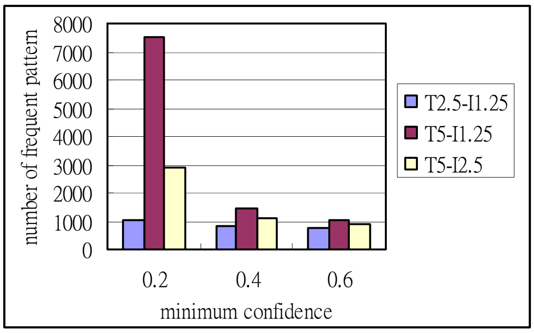

Fig. 10 Comparison of Number of Patterns by Average Number of Items in a Transaction and Number of Items in an Association Rule

V. Conclusions

We proposed a new CAI-PrefixSpan algorithm to discover timed sequential patterns. We apply the confidence concept of association rules to filter the timed sequential patterns, so that the decision makers could be confident about the possibility of an event happening within certain time interval. With the CAI-PrefixSpan algorithm, users could specify a looser minimal support requirement so that important and reliable sequential patterns would be discovered. The timed sequential patterns proposed supports the "within" relationship between shorter and longer time intervals. With the introduction of minimal confidence constraint, the performance of CAI-PrefixSpan is better than the PrefixSpan and I-PrefixSpan.

Inheriting the advantage of PrefixSpan, CAI-PrefixSpan only needs to scan the database once. Additionally, through the recursive pattern growth, the requirement of confidence is checked in each extending. Therefore, when the support is low, CAI-PrefixSpan could eliminate huge numbers of sequential patterns with no reliable possibility of happening. When the database size is large, CAI-PrefixSpan may suffer from the shortage of memory. How to use distributed processing to ease up the memory burden would be an interesting future research topic. How to handle timing issues, like combination of the "simultaneity" and "containment" relationships, would need more investigation. Finally, applications of these timing related data mining algorithms in real cases are interesting.

REFERENCES

[1] Agrawal, R., and Srikant, R., 1994, Fast algorithms for mining association rules, Proc. of International Conference on Very Large Data Bases Conference, pp. 487–499.

[2] Agrawal, R., and Srikant, R., 1995, Mining sequential patterns, Proc. of International Conference on Data Engineering (ICDE'95), pp. 3–14.

[3] Chen, Y.L., Chiang, M.C., Ko, M.T., 2003. Discovering time-interval sequential patterns in sequence databases. Expert Systems with Applications 25, 343–354.

[4] Fiot, C., Laurent,A., Teisseire, M., 2007, Extended Time Constraints for Sequence Mining, Proc. of the 14th International Symposium on Temporal Representation and Reasoning, pp. 105—116.

[5] Guyet, T., and Quiniou, R., 2011, Extracting Temporal Patterns from Interval-Based Sequences, Proc. of the 22nd International Joint Conference on Artificial Intelligence, pp. 1306-1311.

[6] Han, J., Pei, J., Yin, Y., 2000. Mining frequent patterns without candidate generation, Proc. of International Conference on Management of Data, pp. 1–12.

[7] Masseglia, F., Poncelet, P., Teisseire, M., 2009. Efficient mining of sequential patterns with time constraints: reducing the combinations. Expert Systems with Applications 36 (2), 2677–2690.

[8] Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C., 2004. Mining sequential patterns by pattern-growth: the prefixspan approach. IEEE Transactions on Knowledge and Data Engineering 16, 1424–1440.

[9] Shyur,H., Jou,C., Chang, K., 2013. A data mining approach to discovering reliable sequential patterns. Journal of Systems and Software 86(8), 2196–2203.

[10] Srikant, R., Agrawal, R., 1996. Mining sequential patterns: generalizations and performance improvements, Proceedings of the 5th International Conference on Extending Database Technology, pp. 3–17.

[11] Yang, K., Hong, T., Chen, Y., Lan, G., 2013. Projection-based partial periodic pattern mining for event sequences, Expert Systems with Applications 40, p. 4231—p.4240.