Incremental Mining of Closed Sequential Patterns in Multiple Data Streams

Shih-Yang Yang
Department of Media Art, Kang-Ning Junior College of Medical Care and Management, Taipei, Taiwan 114, R.O.C.
Email: shihyang@knjc.edu.tw

Ching-Ming Chao
Department of Computer Science and Information Management, Soochow University, Taipei, Taiwan 100, R.O.C.
Email: chao@csim.scu.edu.tw

Po-Zung Chen and Chu-Hao Sun
Department of Computer Science and Information Engineering Tamkang University, Tamsui, Taiwan 25137, R.O.C.
Email: pozung@mail.tku.edu.tw, 894190320@s94.tku.edu.tw

Abstract—Sequential pattern mining searches for the relative sequence of events, allowing users to make predictions on discovered sequential patterns. Due to drastically advanced information technology over recent years, data have rapidly changed, growth in data amount has exploded and real-time demand is increasing, leading to the data stream environment. Data in this environment cannot be fully stored and ineptitude in traditional mining techniques has led to the emergence of data stream mining technology. Multiple data streams are a branch of the data stream environment. The MILE algorithm cannot preserve previously mined sequential patterns when new data are entered because of the concept of one-time fashion mining. To address this problem, we propose the ICspan algorithm to continue mining sequential patterns through an incremental approach and to acquire a more accurate mining result. In addition, due to the algorithm constraint in closed sequential patterns mining, the generation and records for sequential patterns will be reduced, leading to a decrease of memory usage and to an effective increase of execution efficiency.

Index Terms—Multiple Data Streams, Data Stream Mining, Sequential Pattern Mining, Incremental Mining

I. INTRODUCTION

Due to widespread use of the network, increasing hardware processing speed, and expanding disk storage capacity, the amount of data stored on the computer and network is expanding enormously. As such, discovering useful knowledge from large amounts of data is of particular importance for businesses and relevant practitioners. The objective of data mining is merely to discover knowledge from large amounts of data, which enables the businesses to understand and predict user behaviors so as to expand business opportunities.

Various data mining techniques have been developed to meet different needs, while sequential pattern mining is one of the techniques. Sequential pattern mining is to discover sequential patterns that frequently occur in time sequence or specific order. Through analysis on the state change of sequences, we can make predictions on future states. For example, physicians can discover the evolution process of diseases in terms of patients’ medical records in order to prevent and cure diseases sooner.

Traditionally, data are first stored in databases or integrated into data warehouses and are then provided for data mining. However, the advancement of computer and network technology leads to the emergence of the data stream environment, which is data rapid growing, information fast changing, and real-time demand highly enhancing. In such an environment, stream data are different from traditional data in which they are changing rapidly, massively or possibly infinitely, and fail to be completely stored. Therefore, traditional data mining techniques were ineffective. As a result, new approaches to data stream mining have been studied, in which [1, 2, 3, 4] proposed techniques for mining sequential patterns over data streams.

In recent years, an increasing number of emerging applications start off to gear towards monitoring multiple data streams in order to perform more advanced analysis. In the intensive care unit (ICU), for example, medical devices used on a patient will generate multiple data streams. By analyzing these stream data, doctors can receive a more insightful understanding of the patient’s physiological changes. Therefore, mining in multiple data streams is one of the latest research issues for data mining.

Among the studies on sequential pattern mining in multiple data streams, Oates and Cohen [5] proposed the MSDD (Multi-Stream Dependency Detection) algorithm to search from different data streams for the rules of particular events taking place at a fixed time frame. Chen et al. [6] held that the rules found by MSDD merely account for an exception of sequential patterns and therefore proposed the MILE (Mining in Multiple Streams) algorithm to search for complete sequential patterns. However, MILE uses one-time fashion for mining multiple data streams at a certain time period. When new stream data are input, therefore, MILE only mines new data rather than integrates new mining results with old ones to generate more accurate sequential patterns.
In view of the aforementioned problems, this paper proposes ICspan (Incremental Mining of Closed Sequential Patterns in Multiple Data Streams) algorithm for mining sequential patterns through an incremental approach, in order to provide a more accurate mining result and to reduce special and time consumption in the mining process. Nonetheless, we have the following difficulties to overcome during the study process. First, the data amount in multiple data streams environment is more enormously relative to a single data stream, while the more sequential patterns will also be generated. Under the limitation for memory capacity, an effective processing for unnecessary sequential patterns is required. As a result, in the data stream environment, data could not be stored indefinitely, and to assure continuous data entry, preserve the accuracy of sequential pattern mining results.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 presents the SPAMDAS method, which covers data sampling and incremental mining. Section 4 evaluates and compares the performance of the ICspan algorithm. Section 5 concludes this paper.

II. RELATED WORK

Due to the rising data stream environment in recent years, many studies have started to improve traditional data mining algorithm to accommodate data stream environment. Mining techniques for data stream environment have been put forward in Association Rule, Classification and Clustering, as well as sequential pattern mining.

First of all, the elSeq method proposed by Chang and Lee [1] will be discussed. This method could catch the latest change for the sequential data stream within a short time frame; while through a decaying mechanism to gracefully discarding the non-useful old data.

In the same year, Chen et al. [6] propose the MILE algorithm to implement weaknesses in the two papers [5, 7], which was used to process multiple data streams within a time cycle. In the time-series data stream environment, input data are considered a series of consecutive and time ordering token. Tokens include stream identification number, timepoint, and values, while each data stream is composed by tokens from the identical stream identification number. For example, in the stock market, stream identification number may serve as a stock identification, while in the medical treatment; stream identification number could represent the model number of a medical instrument. In addition, all tokens for multiple data streams at the same time point are all synchronously recorded, and the sequential pattern mining across data stream will be conducted in response to these multiple data streams in synchronized records. While MILE is a reformed method based on PrefixSpan, enhancing efficiency for algorithm through recording already generated sequential patterns and expediting search by a hash method. However, the algorithm records generated sequences based on the fundamental framework of PrefixSpan, accordingly resulting in a high consumption of memory. In addition, due to MILE is a one-time fashion mining algorithm, previously mining results could not be preserved.

Subsequently, Raissi et al. [4] proposed SPEED (Sequential Patterns Efficient Extraction in Data Streams) algorithm to search for the maximal sequential pattern in the data stream. This algorithm maintains frequent sequential patterns based on the new data structure in addition to augmenting prompt decaying strategies, allowing users to search for the maximal sequential pattern at an arbitrary time interval in any given time.

Nonetheless, Ho et al. [3] proposed IncSPAM (Incremental Mining of Sequential Patterns using SPAM) algorithm which differs from earlier mentioned methods in which it applies incremental method for mining sequential patterns from the data stream. In this algorithm, bitmap representation is used for calculating the supports of sequential patterns, in addition to implementation through the concept of CBASW (Customer Bit-Vector Array with Sliding Window), in order to raise the supports for calculating sequential patterns and the speed of sequential pattern ordering. Furthermore, to process the false positive problems from out-of-date data, the concept of decay-rate proposed by Chang and Lee [8] will be improved and used to determine the importance of data through a decay mechanism, allowing algorithms to accommodate to the concept drift problems frequently seen for data stream mining.

III. THE SPAMDAS METHOD

From the previous section for literature review, we have discovered the relevant studies on sequential pattern
mining in multiple data streams could not preserve the results of the previously mined sequential pattern, leading to a less accurate mining result. To approach such a problem, this paper proposes SPAMDAS (Sequential Pattern Mining in Multiple Data Streams) method to solve this problem. SPAMDAS mainly consist of two phases in data sampling and incremental mining. First, in the data sampling phase, a sliding window mode samples from stream data and to divide this sample data into units in lieu of the basic window. Followed by the incremental mining phase, the different basic windows in the sliding window are matched to search for closed sequential patterns, and further combine and update on the new and old sequential patterns to generate more accurate mining results of sequential patterns. Figure 1 shows the overall process of SPAMDAS.

### A. Data Sampling

As referred in Figure 2, in the data sampling phase, we adopt time-sensitive sliding window to proceed with non-overlapping sampling on input stream data. In addition, we obtain mined periodical sequential patterns, limiting frequent sequential patterns with time interval, less than the fixed time intervals. Therefore, we divide the data within the sliding window according to the scale of time interval into several identical length window (these windows are called basic windows). Then we will be able to match the sequential data from these basic windows to discover a certain period of frequent sequential patterns.

![Figure 2. Example of Sliding Window Sampling.](image)

Figure 2 shows sampling from one sliding window, whereas T1, T2, ..., T9 in the figure represents a time-point respectively. In this figure, sliding window sample data on every 9 timepoint, then displace another 9 timepoints to be next data sampling. We take three time intervals as the periodical length for sequential pattern while dividing the timepoint of sliding window in three equal lengths, in anticipation of discovering frequent sequential pattern through matching different basic windows. Consequently, in the process of mining a sliding window, we first take a basic window as transactions of one customer, while the data at the same timepoint will be treated as transactional data, as showed in Figure 3. In Figure 2, the three lattices, T1, T2, and T3 correspond with three timepoints, T1, T2, and T3 in Figure 3, while the center left S1, S2, and S3 respectively represents a different stream. Each timepoint T in the basic window is a set composed of input values from different streams, equal to the previous set we took as a transaction of one customer. In addition, the three timepoints in the basic window represent the time ordering. With these orderings joined with data at the timepoint, we regarded the basic window as transactions of one customer. For example, in Figure 3, the sequence data in the basic window represent as<(11,33,1) (55,86,81) (7,8,22)> as an example. Thereupon through matching the sequence data of different basic windows in the same sliding window, frequent sequence patterns will be searched. We will set the minimum support = 100%, and therefore the frequent sequence pattern discovered in this sliding window will show <(11,33) (22)>.

![Figure 3. A Basic Window Model.](image)

### B. Incremental Mining

In addition to sampling through sliding window models, mining accurate sequential pattern in multiple data streams environments will tackle the stream data which cannot be preserved permanently, leading to a loss of accuracy. As such, the concept of an incremental mining is introduced. We could take the previously sampled data as the previous database (represented as SW), and on that occasion of a new entry from a sliding window data sampling (represented as SW'), SW' will be processed with sequential pattern mining in an incremental approach in order to be integrated with the result of SW. Nevertheless, due to the massive data for multiple data streams and to avoid excessive branches and nodes generated by the lexicographical sequence tree in the process of sequential pattern mining, which could ultimately lead to problems in memory space consumption and time for searching the lexicographical sequence tree. Also, we have integrated concept of the mining closed sequence pattern, and concurrently using a hash table to index closed sequential patterns and to enhance efficiency for establishment and maintaining of the lexicographical sequence tree.

**Definition 1 (The support of sequential pattern):** Have α be a sequential pattern. The support of α is equal to the transactions of including α divided by all of transactions.

**Definition 2 (Frequent sequential pattern):** Have α be a sequential pattern. α is a frequent sequential pattern if and merely if the support of α is larger or equal to minimum support.

**Definition 3 (Semi-frequent sequential pattern):** Have α be a sequential pattern. α is semi-frequent sequential pattern if and merely if the support of α is smaller than minimum support and larger or equal to buffer ratio r*minimum support.

**Definition 4 (Frequent closed sequential pattern):** Have α be a sequential pattern. α is a frequent closed sequential pattern if and merely if α is a frequent sequential pattern...
and there exists no proper super sequential pattern which has the same support as \( \alpha \).

Definition 5 (Semi-frequent closed sequential pattern): Have \( \alpha \) be a sequential pattern. \( \alpha \) is a semi-frequent closed sequential pattern if and merely if \( \alpha \) is a semi-frequent sequential pattern and there exists no proper super sequential pattern which has the same support as \( \alpha \).

In the incrementally update sequential patterns, processing will occur in new entry data, influencing the original sequential pattern mining result, and possibly leading to the following situations:

1. Frequent closed sequential pattern remains the same.
2. Frequent closed sequential pattern becomes a semi-frequent closed sequential pattern.
3. Semi-frequent closed sequential pattern remains the same.
4. Semi-frequent closed sequential pattern becomes a frequent closed sequential pattern.
5. Frequent closed sequential pattern becomes a non-frequent closed sequential pattern.
6. Semi-frequent closed sequential pattern becomes a non-frequent closed sequential pattern.
7. New data bring on new item.
8. Non-frequent closed sequential pattern becomes a frequent closed sequential pattern.
9. Non-frequent closed sequential pattern becomes a semi-frequent closed sequential pattern.

From Situation 1 to 4, due to our preservation of sequential pattern information from the lexicographical sequence tree, we could modify the preserved information directly for any update in new data. In Situation 5 and 6, the support of sequential pattern is less than the support of semi-frequent sequential pattern, will delete the sequential pattern information from the lexicographical sequence tree. In Situation 7 where new data bringing new entry, in other words, the sequence data non-exist in the lexicographical sequence tree, consequently will find out the new data when scanned. In Situation 8, we notice that for a non-frequent closed sequential pattern to become a frequent closed sequential pattern, all of the previously existing sub-sequential patterns of this sequential pattern must be a frequent sequential pattern and without any ultra sequential pattern. Simply put, if one of the sub-sequential pattern is non-frequent in the lexicographical sequence tree then this sequential pattern does exist. Situation 9 is similar to Situation 8 whereas for a non-frequent closed sequential pattern to become a semi-frequent closed sequential pattern, all of the previously existing sub-sequential patterns in the lexicographical sequence tree must be frequent or semi-frequent closed sequential patterns without containing any super sequence. Otherwise, the sequence will not be accounted for a semi-frequent closed sequential pattern.

Figure 4 shows the parameter definitions for the incremental closed sequential pattern mining algorithm (as known as ICspan algorithm) as in the following:

- \( SW' \) is the data for the current sliding window, whereas \( SW \) is the data of the previous sliding window.
- \( r \) is the buffer ratio (0 \( r \) 1), used for determining the number of semi-frequent closed sequential patterns.
- \( \min\_sup \) is the minimum support.
- \( F \) is the frequent closed sequential pattern in \( SW \), whereas \( F' \) is the newly found frequent closed sequential pattern.
- \( SF \) is the semi-frequent closed sequential pattern found in \( SW \), whereas \( SF' \) is the newly discovered semi-frequent closed sequential pattern.
- \( \sup_{sw}(p) \) is the support for sequential pattern \( p \) in \( SW \).
- \( \sup_{sw}(p) \) is the support for sequential pattern \( p \) in \( SW' \).
- \( \sup(p) \) is the support for sequential pattern \( p \).
- \( \supappend(p) \) is the support for sequential pattern \( p \) in the sequential set of all appended items or itemsets.

![Algorithm: ICspan](image)

The steps for ICspan algorithm are divided into two sections. The first section is to identify a new sequential pattern (line 1–4). This section is used to find the closed sequential pattern from the new data and therefore the sequential pattern with length = 1 will be projected. The second section is an integration of the new and old sequential patterns (line 5–13). This section projects on the new patterns found in Section 1 with the previously recorded patterns in order to find the combined sequential patterns derived from the new and old items. The following are the detailed steps to ICspan algorithm:

Step 1 (line 1): Set \( F' \) and \( SF' \) as empty sets.

1. \( F' \leftarrow \psi; SF' \leftarrow \psi; \)
2. Scan \( SW' \) to find 1-item sequential patterns and add into \( F' \) or \( SF' \);
3. For each 1-item sequential pattern in \( F' \) or \( SF' \) do
   - transform into closed sequential patterns and add into \( F' \) or \( SF' \);
4. For each sequential pattern \( p \) in \( F \) or \( SF \) do
   - If \( \sup_{sw}(p) + \sup_{sw}(p) \geq \min\_sup \) then
     - If \( p \) is a closed sequential pattern then add \( p \) into \( F' \);
     - If \( \supappend(p) \leq r*\min\_sup \) then
       - transform into closed sequential patterns and add into \( F' \) or \( SF' \);
   - If \( \sup(p) + \sup(p) \leq \min\_sup \) \( \text{and} \ \sup_{sw}(p) + \sup_{sw}(p) \geq r*\min\_sup \) then
     - If \( p \) is a closed sequential pattern then add \( p \) into \( SF' \);
5. Return;
Step 2 (line 2): Scan the data sampling from the sliding window, find a sequential pattern with length = 1 and support larger or equal to min_sup or r*min_sup. Add the result to F’ or SF’ respectively.

Step 3 (line 3-4): Project the all of the new sequential patterns in F’ or SF’ with length = 1, then transform into the closed sequential patterns.

Step 4 (line 5-11): Previously recorded frequent or semi-frequent sequential patterns are updated and projected then transformed into closed sequential patterns.

Step 4.1 (line 6): Determine if the support for sequential pattern p in SW added with support in SW’ is larger or equal to the minimum support. If the support is determined to true then it becomes a frequent sequential pattern and executes steps in 4.2 and 4.3.

Step 4.2 (line 7): Determine if sequential pattern p includes other sequential patterns or is included under other sequential pattern. Update to assure recorded sequential patterns are frequent closed sequential patterns.

Step 4.3 (line 8-9): Determine if the support for sequential pattern p in all appended items or item sets in the sequential pattern set is larger or equal to (1-r)*min_sup. If the support is determined to be true, then it is possible that a non-frequent sequential pattern has becomes a frequent sequential pattern, which requires projection in order to generate closed sequential patterns.

Step 4.4 (line 10-11): Determine if the support for sequential pattern p in SW added with support in SW’ is smaller minimum support and larger or equal to the r*min_sup. If the support is determined to true then this sequential pattern p is a semi-frequent sequential pattern and requires update to assure the recorded sequential patterns are all semi-frequent closed sequential patterns.

Step 5 (line 12): Store F’ back to F; store SF’ back to SF. Step 6 (line 13): Output F and SF.

The following example will be used to explain the ICspan algorithm. First, we will set min_sup = 100% and r = 0.5. Figure 5 will show data sampling from the first execution of ICspan. Step 1 will set the two empty sets, F’ and SF’, to store the newly found sequential pattern. Step 2 searches for new data sampling from the sliding window which are frequent or semi-frequent sequential patterns with length = 1. The frequent sequential patterns found in the example of figure 5 are <A>, <a> and <3>. The semi-frequent sequential patterns found in the example are <6>, <D> and <6>, then add each of the found sequential patterns to F’ or SF’. Step 3 will project the all of new sequential patterns in F’ or SF’ with length = 1 and transform into frequent closed sequential patterns <(A, a)(3)>, <(A, a)(F)>, and <(A, a)(d, 3)>. Due to lack of previously mined sequential patterns in Step 4, the step will go straight to Step 5 to update F and SF. Lastly, Step 6 output F and SF. Table I shows the result of the first ICspan execution.

<table>
<thead>
<tr>
<th>Time Stream</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
<th>T8</th>
<th>T9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>A</td>
<td>C</td>
<td>F</td>
<td>A</td>
<td>E</td>
<td>B</td>
<td>A</td>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td>S2</td>
<td>a</td>
<td>b</td>
<td>d</td>
<td>a</td>
<td>c</td>
<td>f</td>
<td>a</td>
<td>e</td>
<td>D</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

**Figure 5. Data sampling of the first ICspan execution.**

<table>
<thead>
<tr>
<th>Frequent closed sequential pattern F</th>
<th>&lt;(A, a)(3)&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semi-frequent closed sequential pattern SF</td>
<td>&lt;(6)(3)&gt;, &lt;(A, a)(F)&gt;, &lt;(A, a)(d, 3)&gt;</td>
</tr>
</tbody>
</table>

**TABLE I. RESULT OF FIRST ICSPAN EXECUTION.**

<table>
<thead>
<tr>
<th>Time Stream</th>
<th>T10</th>
<th>T11</th>
<th>T12</th>
<th>T13</th>
<th>T14</th>
<th>T15</th>
<th>T16</th>
<th>T17</th>
<th>T18</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>C</td>
<td>A</td>
<td>E</td>
<td>B</td>
<td>D</td>
<td>B</td>
<td>A</td>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td>S2</td>
<td>f</td>
<td>c</td>
<td>b</td>
<td>a</td>
<td>d</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>S3</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

**Figure 6. Data sampling of second ICspan execution.**

Figure 6 shows data sampling of the second ICspan execution. Step 1-2 will find frequent sequential patterns <3> and semi-frequent sequential patterns <A>, <6>, <a>, <D>, <F> and <6> with length = 1 from figure 6, and then add <3> to F’ while adding the remaining sequential patterns to SF’. Step 3 will project the all of new sequential patterns in F’ or SF’ with length = 1 and transform into semi-frequent closed sequential patterns <(A, 3)(6)>.

Table II shows the results for F’ and SF’ to Step 3. Step 4 combines new and old sequential patterns. First, Step 4.1 determines the support for sequential patterns in Table I added with support for the frequent or semi-frequent sequential patterns from Table II is larger or equal to min_sup. If the condition is in accordance to Step 4.2 will proceed projecting and transformed into frequent closed sequential patterns <3>. Furthermore, since the support of these sequential patterns for all appended sequential sets in Step 4.3 is smaller then (1-r)*min_sup, the projection will not take place to transform the new frequent or semi-frequent closed sequential patterns. Table III shows the result for F’ and SF’ to Step 4.3. Next, Step 4.4 will be determined if the support for sequential pattern p in SW added with support in SW’ is smaller minimum support and larger or equal to the r*min_sup, and this step will add <A>, <a>, <6> · <(A, a)(3)> to SF’. Step 5 will store the last result of F’ and SF’ back to F and SF, then output F and SF through Step 6. Table IV shows the final output result.
IV. PERFORMANCE EVALUATION

A. Experimental Environment and Data

The program has been written by C++ STL in the Visual Studio 2005 compiling environment. All experiments have been conducted in the experimental environment described in Table V. We used the data generator from SQL Server 2005 to generate synthetic datasheets a Uniform distribution and Gaussian distribution. Table VI describes the generated parameters of the synthetic data. For example, S9T200V4 is expressed by 9 stream numbers and 200 data at a timepoint, with the value range for each data stream as much as 4 times more from the multiple data streams environment.

<table>
<thead>
<tr>
<th>Item</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Pentium4 3.00GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>1 GB</td>
</tr>
<tr>
<td>Hard Drive</td>
<td>80 GB</td>
</tr>
<tr>
<td>O.S</td>
<td>WINDOWS XP</td>
</tr>
<tr>
<td>Programming Language</td>
<td>C++ STL</td>
</tr>
</tbody>
</table>

B. Comparison Between ICspan and MILE

In multiple data stream environments, MILE algorithm emphasizes mining sequential patterns from periodical data through recoding generated sequential patterns and using a hash method to shorten the time to mine sequential patterns. Therefore the process does not involve an incremental mining. In addition, due to the massive amount of generated data from multiple data streams, the recorded sequential patterns are also relatively increasing as a result. On the contrary, ICspan algorithm not only conducts incremental mining but also reduces the number of recorded sequential patterns.

(1) Accuracy

From the mining results of sequential patterns in an identical data environment, if the new sequential pattern $\beta$ includes the previous sequential pattern $\alpha$ [9], and then it means $\beta$ has a longer and more accurate length than that of $\alpha$. Through comparing the derivations of mining sequential pattern set from ICspan and MILE, we will observe that all the sequential pattern sets of MILE corresponding to those derived from ICspan. For this reason, we use the following formula to calculate their accuracy:

\[ \sum_{i=1}^{N} p_i \quad (1) \]

\[ \sum_{i=1}^{N} p_i' \quad (2) \]

$N$ is the corresponding sequential pattern number for both algorithms in individual mining result, whereas $p_i$ ($1 \leq i \leq N$) represents the sequential pattern length of each MILE mining, and $p_i'$ ($1 \leq i \leq N$) represents the corresponding sequential pattern length for the sequential pattern set derived from each ICspan algorithm. By adding the total length of sequential patterns, the higher the value means better accuracy.

Figure 7 shows the comparison of accuracy for ICspan and MILE. The vertical axis from the figure indicates the total amount of the sequential pattern length whereas the horizontal axis indicates the minimum support is represented by percentage. The MILE line represents the total amount of sequential pattern lengths found by MILE, whereas ICspan line represents the sequential pattern set derived from ICspan with the sequential pattern lengths corresponding to MILE. Figure 7 shows under various minimum supports, the total amount of sequential pattern lengths found by ICspan is larger than that of MILE. Since ICspan integrates the results of the new and old sequential pattern mining by using a incremental method, the mining result of ICspan will be more accurate.
(2) Number of Sequential Patterns

Figure 8 shows the comparison of the number of generated sequential patterns for MILE and ICspan in S9T2000V4 environment. The min_sup from the figure indicates the minimum support represented by percentage, while sequential patterns represent the number of generated sequential patterns. In order to compare with MILE, the buffering parameter r is set to 1 for ICspan algorithm. Under various minimum supports, the recorded sequential pattern amount for ICspan is much less than that of MILE. The reason for the difference lies on ICspan looking for closed sequential patterns only, leading to the reduction of many undesired sequential patterns.

![Figure 8. Comparison of the Number of Sequential Patterns.](image)

C. Analysis on ICspan Algorithm

The experiment from the second part emphasizes on the ICspan algorithm in different environments and parameter testing by applying different conditions such as data sizes, basic window lengths and data distribution to interpret the executing efficacy of the algorithm.

(1) Data Size

Figure 9 and 10 show the three different multiple data streams environments in S9T20000V4, S9T2000V4 and S9T200V4, compared with the execution time and memory usage for ICspan. The vertical axis in Figure 9 is the executing time in units of seconds, whereas the vertical axis in Figure 13 is the maximum usage for memory in units of kilobytes (KB). As we observe from Figure 9 and 10, when the minimum support for ICspan is lower than 7%, the execution efficacy is worse. Nonetheless, the execution efficacy to maintain minimum support is satisfying. The reason could result from the number of sequential patterns meeting the conditions, which is excessively enormous, extending the time to transform closed sequential patterns.

![Figure 9. Execution Time for Various Data Size.](image)

(2) Basic window length

Figure 11 shows the comparison of the ICspan execution time for various base window lengths in the multiple data streams environment in S9T2000V4. The line BW = 3 indicates the basic window length is equal to 3, whereas the same indication applies to the other lines. Figure 11 indicates the execution time for BW = 3 is smaller than that of BW = 5 and for BW = 7, whereas BW = 5 has execution time slightly higher than BW = 7 at the minimum support of 8%. As the base window length increases for the former, the sequential patterns also increase the mining time; while due to the change of the basic window length for the latter, shortening the time to transform from sequential data to closed sequential patterns.

![Figure 11. Basic Window Length.](image)

(3) Data distribution

Figure 12 shows the comparison of the execution time in Gaussian distribution and Uniform distribution for executing ICspan, in the multiple data streams environment in S9T2000V4. The figure indicates that the execution time is acceptable when the minimum support is larger than or equal to 25%, while the execution time for Gaussian distribution will climb rapidly following a continuously declining minimum support, resulting from the variation in probability for Gaussian distribution.

![Figure 12. Comparison of Different Data Distribution.](image)
V. CONCLUSION

Most studies favor the analysis on long-term data rather than short-term data because short-term data is easily interfered by instantaneous events. Long-term data have less possibility to be interfered, and instead it is easier to find out the periodicity of occurring events, allowing people to be readily interpreting the periodical changes to events. For this reason, in order to find out the sequential pattern mining resulting from long-term data, we have put forward this ICspan algorithm, applying an incremental mining method to conduct a direct sequential mining to the new data, and then to integrate with former mining results for generating new sequential patterns. In addition, we have implemented the method of closed sequential pattern mining to reduce the number of sequential pattern records. This concept will reduce a waste of memory space in increasing sequential patterns. Finally, the ICspan algorithm is collaborated with sliding window mode to obtain a comprehensive sampling of all historical data, resulting in more accurate sequential patterns for reference and decision-making to all users. The experiment results also support that the ICspan algorithm will effectively reduce the sequential pattern records and consequently reducing the memory usage, while maintaining a sound mining efficiency under continuous data entries.

ACKNOWLEDGMENT

The authors would like to express their appreciation for the financial support from the National Science Council of Republic of China under Project No. NSC 98-2221-E-031-003.

REFERENCES


Shih-Yang Yang received his Ph.D. degree in Computer Science and Information Engineering from Tamkang University, Taiwan, in January 2008. Since January 2008, he is an Associate Professor with the Department of Media Art at Kang-Ning Junior College of Medical Care and Management (Taipei, Taiwan). His research interests include parallel & distributed systems, web technology, and multimedia.

Ching-Ming Chao received his Ph.D. degree in Computer Science from The University of Iowa, Iowa City, Iowa, U.S.A. in 1990. From 1990 to 1992, he was an assistant professor in the Department of Computing Sciences at the University of Scranton, Scranton, Pennsylvania, U.S.A. He joined the faculty of the Department of Computer and Information Science at Soochow University in 1992 as an associate professor. From 1992 to 1996, he served as the department chair. Since 2003, he has been a professor in Department of Computer Science and Information Management at Soochow University. His research interests include data mining, data warehousing, database, and web technology.

Po-Zung Chen received his Ph.D. degree in Computer Science from the University of Iowa in December 1989. From November 1989 to May 1990, he was a visiting Assistant Professor at Michigan Technological University (Houghton, Michigan). Since August 1990, he is an Associate Professor with the Department of Computer Science and Information Engineering at Tamkang University (Taipei, Taiwan). His research interests include object-oriented distributed programming, parallel & distributed systems, and simulation & modeling.

Chu-Hao Sun is currently a candidate of Ph.D. student in department of Computer Science and Information Engineering in Tamkang University (Taipei, Taiwan). He received his B.E. and M.E. degree from the same university in 1995 and 1998. His research interests include database management system, parallel processing, web technology, and data mining.