*JSc- EUSL(2008)* **Vol.5 No.1, p 83-95**

ISSN 1391-586X: © 2008 Published by Eastern University, Sri Lanka.

# FAST ALGORITHMS FOR SEQUENCE PATTERN RECOGNITION IN MASSIVE DATASETS

D. Gunaseelan

Department of Information Technology, IBRI College of Technology
IBRI, Sultanate of Oman, dgseela@yahoo.com

## ABSTRACT

*Sequential Pattern Mining is the process of applying data mining techniques to a sequential database for the purpose of discovering the correlation relationships that exist among an ordered list of events. The patterns can be used to focus on the retailing industry, including attached mailing, add-on sales and customer satisfaction. In this paper, I present fast and efficient algorithms called* ***AprioriAllSID and GSPSID*** *for mining sequential patterns that are fundamentally different from known algorithms like* ***AprioriAll and GSP*** *(Generalized Sequential Patterns). The algorithm has been implemented on an experimental basis and its performance studied. The performance study shows that the proposed algorithms have an excellent performance over the best existing algorithms.*

***Keywords:*** Data Mining, Sequential Pattern Mining, AprioriAllSID algorithm, GSPSID algorithm, Data Sequence.

## 1. INTRODUCTION

Data Mining, also known as Knowledge discovery in Databases, has attracted a lot of attention. Because of the progress of data collection tools, large amount of transaction data have been generated, but such data being archived and not used efficiently [4]. Data Mining is the method of discovery of useful information such as rules and previously unknown patterns existing between data items embedded in large databases, which allows more effective utilization of existing data.

The problem of mining sequential patterns in a large database of customer transactions was introduced in [2]. A transaction data typically consists of a customer ID, a transaction ID and a transaction time associated with each transaction and item bought per-transaction. By analyzing these customer transaction data, we can extract the sequential patterns such as "10% of customers who buy both A and B also buy C in the next transaction".

Several algorithms have been proposed to find sequential pattern [2] [22]. An algorithm for finding all sequential patterns, named AprioriAll, was presented in [2]. First, AprioriAll discovers all the set of items (itemset) with a user-defined minimum support (large itemset), where the support is the percentage of customer transactions that contain the itemsets. Second, the database is transformed by replacing the itemsets in each transaction with the set of all large itemsets. Last, it finds the sequential patterns. It is costly to transform the database. In [25], a graph-based algorithm DSG (Direct Sequential Patterns Generation) was presented. DSG constructs an association graph to indicate the associations between items by scanning the database once, and generates the sequential patterns by traversing the graph. Though the disk I/O cost of DSG is very low, the related information may not fit in the memory when the size of the database is large.

In [22], GSP (Generalized Sequential Pattern) algorithm that discovers generalized Sequential Patterns was proposed. GSP finds all the frequent sequences without transforming the database. Besides, some generalized definitions of sequential patterns are introduced in [2] [3]. First, time constraints are introduced. Users often want to specify maximum or minimum time period between adjacent elements. Second, flexible definition of a customer transaction is introduced. It allows a user-defined window-size within which the items can be present. Third, given a user-defined taxonomy (*is-a* hierarchy) over the data items, the generalized sequential patterns, which includes items spanning different levels of the taxonomy, is introduced. All the previous algorithms for discovering sequential patterns are serial algorithms. Finding sequential patterns has to handle a large amount of customer transaction data and requires multiple passes over the database, which requires long computation time. Thus, we introduce efficient algorithms for discovering sequential patterns in a large collection of sequenced data.

In this paper, we consider the new algorithms for mining sequential patterns in sequential environment. All the earlier algorithms are multiple pass over the data whereas in the proposed algorithm, the original database is read only and we introduce a new temporary database $D'$ for the next iterations. After completing the first iteration, we can find the candidate sequence of size-2 using temporary database $D'$. Then we can find the candidate k-size sequences until the candidate sequence or temporary database size is empty. At this stage, the database

size is reduced and the number of candidate sequences is also reduced. This feature is used for finding sequential patterns and also reduced the time complexity. So the proposed methods are more efficient than all other methods like AprioriAll and Generalized Sequential Patterns (GSP).

The rest of this paper is organized as follows: Section 2 describes the problem of mining sequential patterns. In section 3, we propose efficient algorithms namely AprioriAllSID and GSPSID for discovering sequential patterns. Relative performance study is given in section 4. Section 5 concludes the paper.

## 2. SEQUENTIAL PATTERN MINING

### 2.1 Statement of the Problem

The problem of mining sequential patterns can be stated as follows: Let $I = \{i_1, i_2, .., i_m\}$ be a set of *m* distinct attributes, also called items. An itemset is a non-empty unordered collection of items (without loss of generality, we assume that items of an itemset are sorted in increasing order). All items in an itemset are assumed to occur at the same time. A sequence is an ordered list of itemsets. An itemset *i* is denoted as $(i_1, i_2, \ldots, i_k)$, where $i_j$ is an item. An itemset with k items is called a k-itemset. A sequence *s* is denoted as $(s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_q)$, where the sequence element $s_j$ is a $s_j$-itemset. A sequence with k-items ($k = \sum_j |s_j|$) is called a k-sequence. For example, $(B \rightarrow AC)$ is a 3-sequence. An item can occur only once in an itemset, but it can occur multiple times in different itemsets of a sequence.

A sequence $p = (p_1 \rightarrow p_2 \rightarrow \ldots \rightarrow p_n)$ is a subsequence of another sequence $q = (q_1 \rightarrow q_2 \rightarrow \ldots \rightarrow q_n)$, denoted as $p \subseteq q$, if there exist integers $i_1 < i_2 < \ldots < i_n$, such that $p_j \subseteq q_{ij}$ for all $p_j$. For example the sequence $(B \rightarrow AC)$ is a subsequence of $(AB \rightarrow E \rightarrow ACD)$, since the sequence elements $B \subseteq AB$, and $AC \subseteq ACD$. On the other hand the sequence $(AB \rightarrow E)$ is not a subsequence of (ABE), and vice-versa. We say that p is a proper subsequence of q, denoted as $p \subset q$, if $p \subseteq q$ and $p \not\subset q$.

A transaction T has a unique identifier and contains a set of items, i.e., $T \subseteq I$. A customer C has a unique identifier and has associated with it a list of transactions $\{T_1, T_2, \ldots, T_n\}$. We assume that no customer has more than one transaction with the same time-stamp, so that we can use the transaction-time as the transaction identifier. We also assume that the list of customer transactions is stored by the transaction-time. Thus the list of transactions of a customer is itself a sequence $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_n$ called a customer sequence. The database D consists of a number of such customer sequences.

A customer sequence C is said to contain a sequence p, if $p \subseteq q$ i.e., p is a subsequence of the customer sequence C. The support or frequency of a sequence C is denoted as $\sigma(p)$, which is the total number of customers that contains this sequence. Given a user-specified threshold called *minimum support* (denoted *min-sup*), we say that a sequence is frequent if it occurs more than minimum support times. The set of frequent k-sequences is denoted as $F_k$. A frequent sequence is maximal if it is not a sub sequence of any other sequence.

The problem of finding sequential patterns can be decomposed into two parts:

i) Generate all combinations of customer sequences with fractional sequence support (i.e., $support_D(C) / |D|$) above a certain threshold called minimum support *m*.

ii) Use the frequent sequences to generate sequential patterns.

The second sub problem is straightforward. However discovering frequent sequences is a non-trivial issue, where the efficiency of an algorithm strongly depends on the size of the candidate sequences.

## 3. AprioriAllSID

In this section we describe the algorithm AprioriAllSID based on [2].

### 3.1 Description

The AprioriAllSID algorithm is shown in figure 1. The feature of the proposed algorithm is that the given customer transaction database D is not used for counting support after the first pass. Rather the set $C'_k$ is used for determining the candidates' sequences before the pass begins. Each member of the set $C_k$ is of the form $< SID, \{ S_k \} >$, where each $S_k$ is a potentially frequent k-sequence present in the sequence with identifier SID. For k=1, $C_1$ corresponds to the database D, although conceptually each sequence i is replaced by the sequence $\{ i \}$. For $k > 1$, $C'_k$ is corresponding to customer sequence S is $< s.SID, \{s \in C_k \mid s \text{ contained in } t \}$ . If s customer sequence does not contain any candidate k-sequence, then $C'_k$ will not have an entry for this customer sequence.

Thus, the number of sequences in the database is greater than the number of entries in $C'_k$. The number of entries in $C'_k$ may be smaller than the number of sequences in database especially for large value of k. In addition, for large values of k, each entry may be smaller

than the corresponding sequence because very few candidate sequences may be contained in the sequence. However, for small values of k, each may be larger than the corresponding sequence because an entry in $C_k$ includes all candidate k-sequences contained in the sequence.

### 3.2 Algorithm AprioriAllSID

In figure 1, we present an efficient algorithm called AprioriAllSID, which is used to discover all sequential patterns in large customer database.

Algorithm AprioriAllSID

1. $L_1$ = {Large size-1 sequences}; // result of L-itemset phase
2. $C'_k$ = database D;
3. For ( k=2; $L_{k-1} = \varnothing$; k++) do begin
4. $C_k$ = New candidate sequences generated from $L_{k-1}$;
5. $C'_k = \varnothing$;
6. for all entries $s \in C'_{k-1}$ do begin
   // determine candidate sequences in $C_k$ contained in the sequence with Identifier s.SID
7. $C_t = \{s \in C_k \mid s\text{-}C[k]) \in$ s.set-of-sequences $\wedge$ (s-C[k]) $\in$ s.set-of-sequences};
8. for each customer sequence C in the database do
9. increment the count of all candidate sequences in $C_k$ that are contained in s;
10. if ($C_t \neq \varnothing$) then $C'_k = C'_k + <$ s.SID, $C_t>$;
11. end;
12. $L_k$ = Candidate sequences in $C_k$ with minimum support;
13. End;
14. Answer = $\cup_k L_k$;

**Figure 1: Algorithm AprioriAllSID**

Procedure Candidate-gen ($L_{k-1}$: frequent (k-1)-itemsets; min-sup: minimum support)

1. For each itemset $l_1 \in L_{k-1}$
2. For each itemset $l_2 \in L_{k-1}$
3. If $(l_1[1] = l_2[2]) \wedge (l_1[2] = l_2[2]) \wedge (l_1[k-1] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$ then
4. $c = l_1 \cup L_2$; // join step: generate candidate sets
5. If has-infrequent-subset (c, $L_{k-1}$) then
6. Delete c; // prune step: remove infrequent candidate sets
7. Else add c to $C_k$;
8. End if;
9. Return;

**Figure 2: Procedure Candidate-gen**

Procedure Has-infrequent-subset(c:candidate k-itemset; $L_{k-1}$: frequent(k-1)-itemset;

1. For each (k-1)-subsets s of c
2. If $s \notin L_{k-1}$ then
3. Return true;
4. Return false;

**Figure 3: Procedure Has-infrequent-subset**

**Example:** Consider the database in figure 4 and assume that minimum support is 2 customer sequences. By using Candidate-gen procedure in figure 2, with size-1 of frequent sequences gives the candidate sequence in $C_2$ by iterating over the entries in $C'_2$ and generates $C'_2$ in step 6 to 11 of figure 1. The first entry in $C'_1$ is < {(1) (5)} {2} {3} {4} > corresponding to customer sequence 10. The $C_t$ at step 7 corresponding to this entry *s* is {{(1) (5)} {2} {3} {4}} are members of s.set-of-sequences.

By using Candidate-gen procedure with $L_2$ gives $C_3$. Making pass over the data with $C'_2$ and $C_3$ generates $C'_3$. This process is repeated until there is no sequence in the customer sequence database.

**Lemma 1:** For all k > 1, if the set of (k-1)-sequences when the SIDs of the generating transactions are kept associated with the candidate $C_{k-1}$ is correct and complete and frequent (k-1) sequence is correct, then the set $C_t$ generated in step 7 in the $k^{th}$ pass is the same as the set of candidate k-sequences in $C_k$ contained in the customer sequence with identifier s.SID.

A candidate sequence s = s[1] ...s[k] is present in the customer sequence s.SID if and only if both $s_1$ = (s-s[k]) and $s_2$ = (s-s[k-1] are in the customer sequence s.SID. Since the candidate k-sequence was found by using Candidate-gen ($L_{k-1}$), all subsequences of $s \in k$ must be frequent. Hence, $s_1$ and $s_2$ must be frequent sequences. Thus, if a candidate sequence $s \in C_k$ is contained in the customer sequence s.SID, $s_1$ and $s_2$ must be members of s.set-of-sequences since $C'_{k-1}$ is complete. A sequence *s* will be a member of $C_t$. Hence, if $c \in C_k$ is not contained in a customer sequence s.SID, s will not be a member of $C_t$.

| Customer Database | |
|---|---|
| TID | Sequence |
| 10 | < { 1 5} {2} {3} {4} > |
| 20 | < {1} {3} {4} {3 5} > |
| 30 | < {1} {2} {3} {4} > |
| 40 | < {1} {3} {5} > |
| 50 | < {4} {5} > |

| $C'_1$ | |
|---|---|
| TID | Set-of-sequences |
| 10 | < { (1) ( 5) } {2} {3} {4} > |
| 20 | < {1} {3} {4} {(3) ( 5)} > |
| 30 | < {1} {2} {3} {4} > |
| 40 | < {1} {3} {5} > |
| 50 | < {4} {5} > |

| $L_1$ | |
|---|---|
| Sequence | Support |
| {1} | 4 |
| {2} | 2 |
| {3} | 4 |
| {4} | 4 |
| {5} | 4 |

| $C_2$ |
|---|
| Itemset |
| {1 2} |
| {1 3} |
| {1 4} |
| {1 5} |
| {2 3} |
| {2 4} |
| {2 5} |
| {3 4} |
| {3 5} |
| {4 5} |

| $C'_2$ | |
|---|---|
| TID | Set-of-sequences |
| 10 | < { {1 2} {1 3}} {1 4} {1 5} {2 3} {2 4} {2 5} {3 4} {3 5} {4 5} > |
| 20 | < {1 3} {1 4} {1 5} {(3 4) (3 5) (4 5) } > |
| 30 | < {1 2} {1 4} {2 3} {2 4} {3 4} > |
| 40 | < {1 3} {1 5} {3 5} > |
| 50 | < {4 5} > |

| $L_2$ | |
|---|---|
| Sequence | Support |
| {1 2} | 2 |
| {1 3} | 4 |
| {1 4} | 3 |
| {1 5} | 3 |
| {2 3} | 2 |
| {2 4} | 2 |
| {3 4} | 3 |
| {3 5} | 2 |
| {4 5} | 2 |

| $C_3$ |
|---|
| Itemset |
| {1 2 3} |
| {1 2 4} |
| {1 3 4} |
| {1 3 5} |
| {1 4 5} |
| {2 3 4} |
| {3 4 5} |

| $C'_3$ | |
|---|---|
| TID | Set-of-sequences |
| 10 | < {{1 2 3}} {1 2 4} {1 3 5} { 1 4 5} {2 3 4} > |
| 20 | < {1 3 4} {1 3 5} {1 4 5} {3 4 5} } > |
| 30 | < {1 2 3} {1 2 4} {1 3 4} {2 3 4} > |
| 40 | < {1 3 5} > |

| $L_3$ | |
|---|---|
| Sequence | Support |
| {1 2 3} | 2 |
| {1 2 4} | 2 |
| {1 3 4} | 3 |
| {1 3 5} | 3 |
| {1 4 5} | 2 |
| {2 3 4} | 2 |

| $C_4$ |
|---|
| Itemset |
| {1 2 3 4} |

| $C'_4$ | |
|---|---|
| TID | Set-of-sequences |
| 10 | < { 1 2 3 4 } {1 2 3 5} > |
| 30 | < {1 2 3 4} > |

| $L_4$ | |
|---|---|
| Sequence | Support |
| {1 2 3 4} | 2 |

**Figure 4: Example**

### 3.3 Algorithm GSPSID

In figure 3, we propose an efficient algorithm called GSPSID, based on [20], which is used to discover all generalized sequential patterns in large customer database.

Algorithm GSPSID

1 Compute $T^*$, a set of ancestor of each item, from taxonomy T.
2 $L_1$ = {Large size-1 sequences}; // Result of Litemset phase.
3 $C'_1$ = database D; k = 2;
4 While ($L_{k-1} = \varnothing$) do Begin
5 $C_k$ = New candidate sequences generated from $L_{k-1}$;
6 If ( k=2) then
7 Delete any candidate sequence in $C_2$ that consists of a sequence of item and its ancestors.
8 Delete any ancestors in $T^*$ that are not present in any of the candidates in $C_k$.
9 $C'_k = \varnothing$;
10 for all entries $s \in C'_{k-1}$ do Begin
// determine candidate sequences in $C_k$ contained in the sequence with Identifier s.SID
11 $C_t$ = {$s \in C_k$ | s-C[k]) $\in$ s.set-of-sequences $\wedge$ (s-C[k]) $\in$ s.set-of-sequences};
12 for each customer sequence s in the database do
13 Add all ancestors of x in $T^*$ to s;
14 Remove any duplicates from s;
15 increment the count of all candidate sequences in $C_k$ that are contained in s;
16 if ($C_t \neq \varnothing$) then $C'_k = C'_k + < s.SID, C_t >$;
17 End;
18 $L_k$ = Candidate sequences in $C_k$ with minimum support;
19 End;
20 Answer = $\cup_k L_k$;

**Figure 5: Algorithm GSPSID**

### 3.4 Description

We add optimizations to GSP algorithm, which gives the algorithm GSPSID. In GSPSID algorithm, given original database D is not used for counting after the first pass. The first pass of algorithm determines the support of each item, like GSP algorithm. At the end of first pass, the algorithm knows which items are frequent, i.e., has minimum support. We introduce

the temporary database D' which is used to determine the candidate sequences before the pass begins. The member of that temporary database is of the form < SID, $\{S_k\}$ >, where each $S_k$ is a potentially frequent k-sequence present in the sequence with identifier SID.

For k=1, $C_1$ is the corresponding temporary database D'. If k=2, then we add three optimizations, based on [2] to reduce the size of the database. If a customer sequence does not contain any candidate k-sequence, then $C'_k$ will not have an entry for this customer sequence. Thus, the number of sequences in the database is greater than the number of entry in $C'_k$. Conversely, the number of entries in $C'_k$ may be smaller than the number of sequences in database especially for large values of k. In addition, for large values of k, each entry may be smaller than the corresponding sequence because very few candidate sequences may be contained in the sequence. For small values of k, each may be larger than the corresponding sequence because an entry in $C_k$ includes all candidate k-sequences contained in the sequence.

### 3.5 Data Structure used

Both algorithms use same data structures. Each candidate sequence is assigned a unique number called its SID. Each set of candidate sequence $C'_k$ is kept in an array indexed by the IDs of the sequences in $C_k$. So, a member of $C'_k$ is of the form SID, {ID} ⟩. Each $C'_k$ is stored in a sequential structure.

***There are two additional fields maintained for each candidate sequence. They are***

1. **Generators**: This field of sequence $C_k$ stores the IDs of the two maximal (k-1) sequence , the combination of which generated $C_k$.
2. **Extensions**: This field stores IDs of all the sequences $C_{k+1}$ obtained as an extension of $C_k$.

Now, s.set-of-sequence of $C'_{k-1}$ gives the IDs of all the (k-1)-candidate sequence contained in transaction s.SID. For each such candidate sequence $C_{k-1}$ the extensions field gives $S_k$ the set of IDs of all the candidate k-sequences that are extensions of $C_{k-1}$. For $C_k$ in $S_k$ the generators field gives the IDs of the two sequences that generated $C_k$. If these sequences are present in the entry for s.set-of-sequences, $C_k$ is present in customer sequence s.SID. Hence we add $C_k$ to $C_t$. By using this data structure we can efficiently store and process the candidate sequences.

## 4. PERFORMANCE EVALUATION

In this section, we describe the experiments and the performance results of AprioriAllSID algorithms. We also compare the performance with the AprioriAll and GSP algorithms. We performed our experiments on an IBM Pentium machine. Using data set generator, we have simulated the data and test algorithms like AprioriAll, AprioriAllSID, GSP and GSPSID. We have used the simulated data for the performance comparison experiments. The data sets are assumed to simulate a customer-buying pattern in a retail environment used in [20].

In the performance comparison, we used the five different data sets. The Table 1 & 2 shows the performance of AprioriAll, GSP, AprioriAllSID and GSPSID for minimum support 1% to 5% for different volume of data. Even though AprioriAllSID and GSPSID seem to be nearly equal, for massive volume of data, the performance of AprioriAllSID and GSPSID will be far better than AprioriAll and GSP algorithms.

**Table 1**: Performance evaluation between AprioriAll and AprioriAllSID algorithms

| DB Size | AprioriAll (Execution Time in Seconds) | | | | | AprioriAllSID (Execution Time in Seconds) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **1%** | **2%** | **3%** | **4%** | **5%** | **1%** | **2%** | **3%** | **4%** | **5%** |
| 100K | 187 | 199 | 211 | 228 | 245 | 98 | 121 | 148 | 164 | 183 |
| 200K | 325 | 339 | 351 | 367 | 384 | 174 | 192 | 221 | 246 | 265 |
| 300K | 428 | 447 | 465 | 489 | 510 | 269 | 281 | 301 | 324 | 356 |
| 400K | 559 | 587 | 611 | 638 | 669 | 346 | 371 | 392 | 415 | 458 |
| 500K | 678 | 691 | 726 | 758 | 793 | 489 | 514 | 561 | 592 | 636 |

**Table 2**: Performance evaluation between GSP and GSPSID algorithms

| DB Size | GSP (Execution Time in Seconds) | | | | | GSPSID (Execution Time in Seconds) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **1%** | **2%** | **3%** | **4%** | **5%** | **1%** | **2%** | **3%** | **4%** | **5%** |
| 100K | 149 | 188 | 213 | 249 | 274 | 67 | 98 | 121 | 149 | 162 |
| 200K | 251 | 289 | 308 | 329 | 368 | 123 | 146 | 178 | 204 | 238 |
| 300K | 339 | 368 | 392 | 421 | 467 | 212 | 258 | 298 | 332 | 378 |
| 400K | 426 | 467 | 493 | 527 | 569 | 320 | 357 | 398 | 435 | 481 |
| 500K | 512 | 541 | 570 | 518 | 536 | 404 | 449 | 481 | 523 | 556 |

Table 1 and 2 show the execution times for the five data sets for an increasing value of minimum support (say 1% to 5%). The execution times increase for both AprioriAllSID and AprioriAll algorithms and GSP and GSPSID as the minimum support is decreased because the total number of candidate sequences increase. AprioriAll algorithm in [2] and GSP [20] are the multiple passes over the data. So, the execution time is increased with increase of the customer transactions in the database. In Table 1 and 2, we can conclude that the AprioriAllSID algorithm is 2 times faster than AprioriAll algorithm and GSP algorithm is 3 times faster than GSPSID for small volume of data and more than the order of magnitude for the large volume of data. The data sets ranges from giga bytes to tera bytes and the proposed algorithms will be much faster than AprioriAll and GSP. Thus we conclude that the proposed algorithms are quite suitable for massive databases.

## 5. CONCLUSION

We present two new algorithms, AprioriAllSID and GSPSID, for discovering all relevant generalized association rules between items in massive database of transactions. We compare AprioriAllSID algorithm with AprioriAll algorithm and GSPSID Algorithm with GSP algorithm in [10]. We presented experimental results, using synthetic data, showing that the proposed algorithms always outperform AprioriAll and GSP algorithms. The performance gap increases with the problem size, and range from the factor of two for small problems to more than an order of magnitude for large problems.

## REFERENCES

[1] R.Agrawal and R. Srikant, Fast Algorithm for Mining Association Rules, *In the Proceedings of the 11th International conference on Very Large Data Bases*, September 1994.

[2] R. Agrawal and R. Srikant, Mining Sequential Patterns, *Journal of Intelligent Systems,* vol. 9. No.1, 1997, pp 33 – 56.

[3] R. Agrawal and R. Srikant, Mining Sequential Patterns, *In the Proceedings of the 11th International Conference on Data Engineering, (ICDE'95)* pp. 3-14, Taipei, Taiwan, March 1995.

[4] R. Agrawal and R. Srikant, Mining Sequential Patterns: Generalization and Performance Improvements, *Research Report RJ 9994, IBM Almaden Research Center*, San Jose, California, December 1995.

[5] S. Cong, J. Han,D.A. Padua. Parallel mining of closed Sequential Patterns, *KDD*, 2005.

[6] M.S. Chen, J.S. Park, and P.S. Yu. Efficient data mining for path traversal patterns. *IEEE Transaction of Knowledge and Data Engineering*, 10(2), March-April 1998.

[7] M. Chen, J. Han, and S. Yu, "Data Mining: An Overview from a Database Perspective", *IEEE Transactions on Knowledge and Data Engineering,* December 1996.

[8] Dunren Che , Wei Zheng, Smart support functions for sequential pattern mining, *Journal of Computational Methods in Sciences and Engineering*, v.6 no.5, 6 Supplement 2, p.255-263, April 2006.

[9] Ezeife, C. and Lu, Y. Mining web log sequential patterns with position coded preorder linked wap-tree. *International Journal of Data Mining and Knowledge Discovery (DMKD)*, Kluwer Publishers, p.5–38, 2005.

[10] M.N. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: Sequential Pattern Mining with regular expression constraints. *In the Proceedings of the. 25th Int'l Conf. on VLDB*, September 1999.

[11] D.Gunaseelan and R. Nadarajan, Fast algorithm for mining Sequential Patterns using Partition Method, *In the Proceeding of an International Conference on Human and Computer*, Aizu, Japan, September 2001.

[12] D. Gunaseelan and R. Nadarajan, Fast algorithm for mining Generalized Association Rules-An Incremental Approach. *In the Proceeding of an International Conference on Information, Decision and Control*, Adeline, South Australia, February 2002.

[13] Valerie Guralnik , George Karypis, Parallel tree-projection-based sequence mining algorithms, *International Journal of Parallel Computing*, v.30 n.4, p.443-472, April 2004.

[14] Chulyun Kim , Jong-Hwa Lim, Raymond T. Ng, Kyuseok Shim, SQUIRE: Sequential pattern mining with quantities, *Journal of Systems and Software*, v.80 n.10, p.1726-1745, October, 2007.

[15] Jia-Wei Han, Jian Pei , Xi-Feng Yan, From sequential pattern mining to structured pattern mining: A pattern-growth approach, *Journal of Computer Science and Technology*, v.19 n.3, p.257-279, May 2004.

[16] Han, J., Pei, J., Yin, Y. and Mao, R. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *International Journal of Data Mining and Knowledge Discovery,* p.53–87, Jan 2004.

[17] J. Han, "Data Mining Techniques", *In the Proceeding of 25th ACM SIGMOD International Conference on Management of Data,* June 1996.

[18] Kuo-Yu Huang , Chia-Hui Chang, Efficient mining of frequent episodes from complex sequences, *Information Systems*, v.33 n.1, pp. 96-114, March, 2008.

[19] S. Parthasarathy, M.J. Zaki, M. Ogihara, and S. Dwarkadas, Incremental and Interactive sequence mining. *In Proceedings of the 1999 ACM 8th international Conference on Informational and knowledge Management (CIKM'99),* Kansas City, MO USA, Nov. 1999.

[20] A. Savasere, E. Omiecinski, and S. Navathe, An Efficient Algorithm for Mining Association Rules in Large Databases, *In the Proceedings of the 21st International Conference on VLDB,* September 1995.

[21] Dhany Saputra, Dayang R.A. Rambli, Oi Mean Foong, Mining Sequential Patterns Using I-Prefix Span, *International Journal of Computer Science and Engineering* 2;2 pp 49-54, 2008

[22] R.Srikant and R. Agrawal. Mining Sequential Patterns: Generalization and Performance Improvements. *In the Proceedings of the 5th International Conference on Extending Database Technology (EDBT'96)*, pages 3-17, Avignon, France, September 1996.

[23] S. Thomas, and S. Sarawagi, Mining generalized association rules and sequential patterns using SQL queries, *In the Proceedings of 4th International Conference on Knowledge Discovery in Databases and Data Mining*, New York, August, 1998.

[24] L.M. Yen and S.Y. Lee, Fast discovery of sequential pattern through memory indexing and database partitioning, *Journal Information Science and Engineering*, vol. 21, pp. 109-128, 2005.

[25] Show-Jane Yen; Chen, A.L.P. An efficient approach to discovering knowledge from large databases, *Fourth International Conference on Parallel and Distributed Information Systems,* 1996, Volume, Issue, December 1996 Page(s):8 – 18.

[26] M.J. Zaki. Efficient enumeration of frequent sequences. *In proceedings of the 1998 ACM 7th International Conference on Information and Knowledge Management (CIKM'98),* Washington, US, 1998.

[27] Zaki, M. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, pp.31–60, 2001.

[28] Baoyao Zhou , Siu Cheung Hui , Alvis Cheuk Ming Fong, Efficient sequential access pattern mining for web recommendations, *International Journal of Knowledge-based and Intelligent Engineering Systems*, v.10 n.2, pp.155-168, April 2006.