International Journal of Computer Science Engineering
and Information Technology Research (IJCSEITR)
ISSN(P): 2249-6831; ISSN(E): 2249-7943
Special Edition, Apr 2015, 61-70

WEB USAGE MINING: EXTRACTION OF SEQUENTIAL PATTERNS USING PREFIXSPAN METHOD

SAHANA V[1] & G T RAJU[2]

[1]Dean of Engginering, Department of Computer Science, RNS Institute of Technology, Bangalore, Karnataka, India

[2]Professor & H.O.D, Department of Computer Science, RNS Institute of Technology, Bangalore, Karnataka, India

ABSTRACT

Sequential pattern mining is an important data mining problem with broad applications. However, it is also a difficult problem since the mining may have to degenerate or examine a combinatorially explosive number of intermediate subsequences. Most of the previously developed sequential pattern mining methods, such as GSP, explore a candidate generation-and-test approach [1] to reduce the number of candidates to be examined. However, this approach may not be efficient in mining large sequence databases having numerous patterns and/or long patterns. In this paper, we propose a projection-based, sequential pattern-growth approach for efficient mining of sequential patterns. In this approach, a sequence database is recursively projected into a set of smaller projected databases, and sequential patterns are grown in each projected database by exploring only locally frequent fragments.

KEYWORDS: Sequential Pattern, Frequent Pattern, Transaction Database, Sequence Database

I. INTRODUCTION

Sequential pattern mining, which discovers frequent subsequences as patterns in a sequence database, is an important data mining problem with broad applications, including the analysis of customer purchase patterns or Web access patterns, the analysis of sequencing or timerelated processes such as scientific experiments, natural disasters, and disease treatments, the analysis of DNA sequences, etc.

The sequential pattern mining problem was first introducedby Agrawal and Srikant in [2]: Given a set of sequences,where each sequence consists of a list of elements and each elementconsists of a set of items, and given a user-specified min_supportthreshold, sequential pattern mining is to find all frequentsubsequences, i.e., the subsequences whose occurrence frequency in the set of sequences is no less than min_support.

The a priori-like sequential pattern mining method, though reducing search space, bears three nontrivial, inherent costs that are independent of detailed implementation techniques.

- A huge set of candidate sequences could be generated in a large sequence database. Since the set of candidate sequences includes all the possible permutations of the elements and repetition of items in a sequence, the apriori-based method may generate a really large set of candidate sequences even for a moderate seed set. For example, two frequent sequences of length-1,⟨a⟩and ⟨b⟩will generate five candidate sequences of length-2: ⟨aa⟩,⟨ab⟩,⟨ba⟩, ⟨bb⟩, and ⟨(ab)⟩, where ⟨(ab)⟩represents that two events, a and b, happen in the same time slot. If there are 1,000 frequent sequences of length-1, such as $\langle a_1\rangle$,$\langle a_2\rangle$,$\langle a_{1000}\rangle$, an a priori-like algorithm will generate 1,000*1,000+(1,000+999)/2=1,499,500 candidate sequences. Notice that the cost of candidate sequence

generation, test, and support counting is inherent to the a priori-based method, no matter what technique is applied to optimize its detailed implementation.

- Multipl e scans of databases in mining. The length of each candidate sequence grows by one at each database scan. In general, to find a sequential pattern of length *l*, the a priori-based method must scan the database at least *l*times. This bears a nontrivial cost when long patterns exist.

- The a priori-based method generates a combinatorially explosive number of candidates when mining long sequential patterns. A long sequential pattern contains a combinatorial explosive number of subsequences, and such subsequences must be generated and tested in the a priori-based mining. Thus, the number of candidate sequences is exponential to the length of the sequential patterns to be mined. For example, let the database contain only one single sequence of length 100, $\langle a_1 a_2 \ldots a_{100} \rangle$, and the min_support threshold be 1 (i.e., every occurring pattern is frequent). To (re)derive this length-100 sequential pattern, the a priori-based method has to generate 100 length-1 candidate sequences (i.e., $\langle a_1 \rangle, \langle a_2 \rangle, \ldots \langle a_{100} \rangle$), 100 *100+(100+99)/2=14,950 length-2 candidate sequences,$\binom{100}{3}$=161,700 length-3 candidate sequences,1 and so on. Obviously, the total number of candidate sequences to be generated is

$\sum_{i=1}^{100} \binom{100}{i} = 2^{100} - 1 \approx 10^3$.

In this paper, we systematically explore a pattern-growthapproach for efficient mining of sequential patterns in largesequence database. The approach adopts a divide-and conquer,pattern-growth principle as follows: Sequence databases are recursively projected into a set of smaller projected databases based on the current sequential pattern(s), and sequential patterns are grown in each projected databases by exploring only locally frequent fragments.

Table 1: A Sequence Database

Sequence_id	Sequence
10	$\langle a(abc)(ac)d(cf) \rangle$
20	$\langle (ad)c(bc)(ae) \rangle$
30	$\langle (ef)(ab)(df)cb \rangle$
40	$\langle eg(af)cbc \rangle$

The remainder of the paper is organized as follows: In Section 2, the sequential pattern mining problem is defined. In Section 3, our approach, projection based sequential pattern growth, is introduced. In section 4, our experimental results are reported.

II. PROBLEM DEFINITION

Let $I = \{i_1, i_2, \ldots, i_n\}$be a set of all items. An itemset is a subset of items. A sequence is an ordered list of item sets. A sequence s is denoted$\langle s_1 s_2 \ldots s_l \rangle$, where s_j is an itemset. $s_j$is also called an element of the sequence, and denoted as $(x_1 x_2 \ldots x_m)$, where x_k is an item. For brevity, the brackets are omitted if an element has only one item, i.e., element (x) is written as$x$. An item can occur at most once in an element of a sequence, but can occur multiple times in different elements of a sequence. The number of instances of items in a sequence is called the length of the sequence. A sequence with length l is called an$l$ −sequence. A sequence $\alpha = \langle a_1 a_2 \ldots a_n \rangle$is called a subsequence of another sequence $\beta = \langle b_1 b_2 \ldots b_m \rangle$ and

β a supersequence of α, denoted as α ⊑ β, if there exist integers $1 \leq j_1 < j_2 < \cdots < j_n \leq m$such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, a_n \subseteq b_{j_n}$.

A sequence database S is a set of tuples$\langle sid, s\rangle$, where sid is a sequence_id and s a sequence. A tuple$\langle sid, s\rangle$ is said to contain sequence α, if α is a subsequence of s. The support of a sequence α in a sequence database S is the number of tuples in the database containing α, i.e.,$support_S(\alpha)$= $|\{\langle sid, s\rangle | (\langle sid, s\rangle \in S) \wedge (\alpha \sqsubseteq s)\}|$. It can be denoted as $support(\alpha)$if the sequence database is clear from the context. Given a positive integer $min\_support$ as the support threshold, a sequence$\alpha$is called a sequential pattern in sequence database$S$if $support_S(\alpha) \geq minsupport$. A sequential pattern with length l is called l −pattern.

Given a sequence database and the $min\_support$ threshold, sequential pattern mining is to find the complete set of sequential patterns in the database.

To avoid checking every possible combination of a potential candidate sequence, one can first fix the order of items within each element. Since items within an element of a sequence can be listed in any order, without loss of generality, one can assume that they are always listed alphabetically. For example, the sequence in S with Sequence_id 10 in our running example is listed as $\langle a(abc)(ac)d(cf)\rangle$ instead of$\langle a(bac)(ca)d(fc)\rangle$. With such a convention, the expression of a sequence is unique. Then, the task is to examine whether one can fix the order of item projection in the generation of a projected database. Intuitively, if one follows the order of the prefix of a sequence and projects only the suffix of a sequence, one can examine in an orderly manner all the possible subsequences and their associated projected database. Thus, it is important to introduce the concept of prefix, suffix and projected database.

III. MINING SEQUENTIAL PATTERNS BY PATTERN GROWTH

Instead of repeatedly scanning the entire database and generating and testing large sets of candidate sequences, one can recursively project a sequence database into a set of smaller databases associated with the set of patterns mined so far and, then, mine locally frequent patterns in each projected database.

PrefixSpan: Prefix-Projected Sequential

Patterns Mining

We examine whether one can fix the order of item projection in the generation of a projected database. Intuitively, if one follows the order of the prefix of a sequence and projects only the suffix of a sequence, one can examine in an orderly manner all the possible subsequences and their associated projected database. Thus, we first introduce the concept of prefix and suffix.

Definition 1 (Prefix)

Suppose all the items within an element are listed alphabetically. Given a sequence

$\alpha = \langle e_1 e_2 \ldots e_n\rangle$ (where each e_i corresponds to a frequent element in S), a sequence $\beta = \langle e_1' e_2' \ldots e_m'\rangle (m \leq n)$ is called a prefix of α if and only if

- $e_1' = e_1$ for $(i \leq m-1)$
- $e_m' \subseteq e_m$

- all the frequent items in($e_m - e'_m$) are alphabetically after those in e'_m

For example,⟨a⟩,⟨aa⟩,⟨a(ab)⟩ and ⟨a(abc)⟩ are prefixes of sequence s⟨a(abc)(ac)d(cf)⟩, but neither ⟨ab⟩ nor ⟨a(bc)⟩ is considered as a prefix if every item in the prefix ⟨a(abc)⟩ of sequence s is frequent in S.

Definition 2 (Suffix)

Given a sequence $\alpha = \langle e_1 e_2 \dots e_n \rangle$ (where each e_i corresponds to a frequent element in S). Let $\beta = \langle e_1 e_2 \dots e_{m-1} e'_m \rangle (m \leq n)$ be the prefix of α. Sequence $\gamma = \langle e''_m e_{m+1} \dots e_n \rangle$is called the suffix of α with regards to prefix β, denoted as $\gamma = \alpha / \beta$, where $e''_m = (e_m - e'_m)$. We also denote $\alpha = \beta.\gamma$.

For example, for the sequence s =⟨a(abc)(ac)d(cf)⟩,⟨(abc)(ac)d(cf)⟩ is the suffix with regards to the prefix⟨a⟩, ⟨(_bc)(ac)d(cf)⟩ is the suffix with regards to the prefix⟨aa⟩, and ⟨(_c(ac)d(cf))⟩ is the suffix with regards to the prefix ⟨a(ab)⟩.

Definition 3(Projected Database)

Let α be a sequential pattern in a sequence database S. The α -projected database, denoted as $S|_\alpha$, is the collection of suffixes of sequences in S with regards to prefix α.

For the same sequence database S in Table 1 with min_sup = 2, sequential patterns in S can be mined by a prefix-projection method in the following steps:

Find Length-1 Sequential Patterns. Scan S once to

Find all the frequent items in sequences. Each of these frequent items is a length-1sequential pattern.

They are ⟨a⟩ : 4,⟨b⟩: 4,⟨c⟩ : 4, ⟨d⟩: 3, ⟨e⟩: 3, and

⟨f⟩ 3, where the notation "⟨pattern⟩ : count" represents

The pattern and its associated support count.

Table 2: Projected Databases and Sequential Patterns

prefix	projected (suffix) database	sequential patterns
⟨*a*⟩	⟨(*abc*)(*ac*)*d*(*cf*)⟩, ⟨(_*d*)*c*(*bc*)(*ae*)⟩, ⟨(_*b*)(*df*)*cb*⟩, ⟨(_*f*)*cbc*⟩	⟨*a*⟩, ⟨*aa*⟩, ⟨*ab*⟩, ⟨*a*(*bc*)⟩, ⟨*a*(*bc*)*a*⟩, ⟨*aba*⟩, ⟨*abc*⟩, ⟨(*ab*)⟩, ⟨(*ab*)*c*⟩, ⟨(*ab*)*d*⟩, ⟨(*ab*)*f*⟩, ⟨(*ab*)*dc*⟩, ⟨*ac*⟩, ⟨*aca*⟩, ⟨*acb*⟩, ⟨*acc*⟩, ⟨*ad*⟩, ⟨*adc*⟩, ⟨*af*⟩
⟨*b*⟩	⟨(_*c*)(*ac*)*d*(*cf*)⟩, ⟨(_*c*)(*ae*)⟩, ⟨(*df*)*cb*⟩, ⟨*c*⟩	⟨*b*⟩, ⟨*ba*⟩, ⟨*bc*⟩, ⟨(*bc*)⟩, ⟨(*bc*)*a*⟩, ⟨*bd*⟩, ⟨*bdc*⟩, ⟨*bf*⟩
⟨*c*⟩	⟨(*ac*)*d*(*cf*)⟩, ⟨(*bc*)(*ae*)⟩, ⟨*b*⟩, ⟨*bc*⟩	⟨*c*⟩, ⟨*ca*⟩, ⟨*cb*⟩, ⟨*cc*⟩
⟨*d*⟩	⟨(*cf*)⟩, ⟨*c*(*bc*)(*ae*)⟩, ⟨(_*f*)*cb*⟩	⟨*d*⟩, ⟨*db*⟩, ⟨*dc*⟩, ⟨*dcb*⟩
⟨*e*⟩	⟨(_*f*)(*ab*)(*df*)*cb*⟩, ⟨(*af*)*cbc*⟩	⟨*e*⟩, ⟨*ea*⟩, ⟨*eab*⟩, ⟨*eac*⟩, ⟨*eacb*⟩, ⟨*eb*⟩, ⟨*ebc*⟩, ⟨*ec*⟩, ⟨*ecb*⟩, ⟨*ef*⟩, ⟨*efb*⟩, ⟨*efc*⟩, ⟨*efcb*⟩.
⟨*f*⟩	⟨(*ab*)(*df*)*cb*⟩, ⟨*cbc*⟩	⟨*f*⟩, ⟨*fb*⟩, ⟨*fbc*⟩, ⟨*fc*⟩, ⟨*fcb*⟩

Divide Search Space

The complete set of sequentialpatterns can be partitioned into the following six subsets according to the six prefixes: 1) the ones with prefix ⟨a⟩, 2) the ones with prefix ⟨b⟩, ., and 3) the ones with prefix ⟨f⟩.

Find Subsets of Sequential Patterns

The subsets of sequential patterns can be mined by constructing the corresponding set of projected databases and mining each recursively. The projected databases as well as sequential patterns found in them are listed in Table 2, while the mining process is explained as follows:

- Find sequential patterns with prefix ⟨a⟩. Only the sequences containing ⟨a⟩ should be collected. Moreover, in a sequence containing ⟨a⟩, only the su bsequence prefixed with the first occurrence of ⟨a⟩ should be considered. For example, in sequence⟨(ef)(ab)(df)(cb)⟩, only the subsequence⟨(_b)(df)(cb)⟩ should be considered for mining sequential patternsprefixed with ⟨a⟩. Notice that (_b) means that the last element in the prefix, which is a, together with b, form one element.

The sequences in S containing ⟨a⟩ are projected with regards to ⟨a⟩ to form the ⟨a⟩projected database, which consists of four suffix sequences:⟨(abc)(ac)d(cf)⟩,⟨(d)c(bc)(ae)⟩,⟨(b)(df)cb⟩, and ⟨(_f)cbc⟩.

By scanning the ⟨a⟩-projected database once, its locally frequent items are a : 2, b : 4, b : 2, c : 4, d : 2, and f : 2. Thus, all thelength-2 sequential patterns prefixed with ⟨a⟩ are found, and they are: ⟨aa⟩ : 2, ⟨ab⟩ : 4, ⟨(ab)⟩ : 2, ⟨ac⟩ : 4, ⟨ad⟩ : 2, and ⟨af⟩ : 2.

Recursively, all sequential patterns with prefix ⟨a⟩ can be partitioned into six subsets:

1)Those prefixed with ⟨aa⟩, 2) those with ⟨ab⟩,. ., and, finally, 3) those with ⟨af⟩. These subsets can be mined by constructing respective projected databases and mining each recursively as follows:

- The ⟨aa⟩-projected database consists of two nonempty (suffix) subsequences prefixed with ⟨aa⟩: ⟨(bc)(ac)d(cf)⟩, {⟨(_e)⟩}. Since there is no hope to generate any frequent subsequence from this projected database, the processing of the ⟨aa⟩projected database terminates.
- The ⟨ab⟩-projected database consists of threesuffix sequences: ⟨(_c) ⟩(ac)d(cf) ⟩, ⟨(_c)a⟩, and ⟨c⟩. Recursively mining the ⟨ab⟩-projected database returns four sequentialpatterns:
- ⟨(_c) ⟩, ⟨(_c)a⟩,⟨a⟩, and ⟨c⟩ (i.e., ⟨a(bc) ⟩, ⟨a(bc)a⟩, ⟨aba⟩, and ⟨abc⟩.) They form the complete set of sequential patterns prefixed with ⟨ab⟩.
- The ⟨(ab)⟩-projected database contains only two sequences: ⟨(_c)(ac)d(cf)⟩ and ⟨(df)cb⟩, which leads to the finding of the following sequential patterns prefixed with⟨(ab)⟩: ⟨c⟩, ⟨d⟩, ⟨f⟩, and ⟨dc⟩.
- The ⟨ac⟩, ⟨ad⟩, and ⟨af⟩-projected databases can be constructed and recursively mined similarly. The sequential patterns found are shown in Table 2.
    - Find sequential patterns with prefix ⟨b⟩, ⟨c⟩, ⟨d⟩, ⟨e⟩, and ⟨f⟩, respectively. This can be done by constructing the ⟨b⟩, ⟨c⟩, ⟨d⟩, ⟨e⟩, and ⟨f⟩projected databases and mining them, respectively. The projected databases as well as the sequential patterns found are shown in Table 2.

The Set Of Sequential Patterns is the Collection of Patterns Found in the above Recursive Mining Process.

Onean verify that it returns exactly the same set of sequential patterns as what GSP and FreeSpan do.

Based on the above discussion, the algorithm of Prefix-Span is presented as follows:

Algorithm (PrefixSpan) Prefix-projected Sequential Pattern mining.

Input: A sequence database S, and the minimum support threshold min_support.

Output: The complete set of sequential patterns.

Method: Call PrefixSpan$(<>, 0, S)$

Subroutine PrefixSpan$(\alpha, l, S|_\alpha)$

The parameters are 1) α is a sequential pattern; 2) l is the length of α; and 3) $S|_\alpha$ is the α -projected database if $\alpha \neq <>$ otherwise, it is the sequence database S.

METHODS

- Scan $S|_\alpha$ once, find each frequent item, b, such that
  - b can be assembled to the last element of α to form a sequential pattern; or
  - ⟨b⟩can be appended to α to form a sequential pattern.

2. For each frequent item b, append it to α to form a sequential pattern α', and output α'.

3. For each α', construct α'-projected database $S|_{\alpha'}$, and call PrefixSpan$(\alpha', l+1, S|_{\alpha'})$.

High level design gives an overview of the logical flow. However this suffices the user to understand the logic. Figure 1 depicts the basic knowledge about the system design and the architecture.

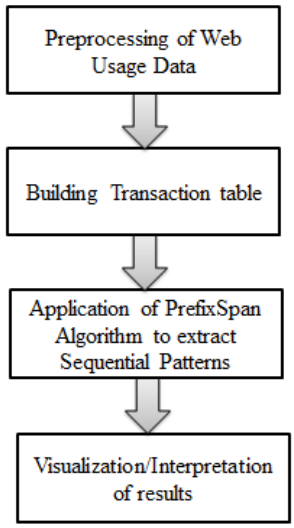

Figure 1: High Level Flow Chart

System Architecture of the proposed work is shown in figure 2. It consists of Preprocessing, Sequential Pattern Mining and Visualization of results modules.

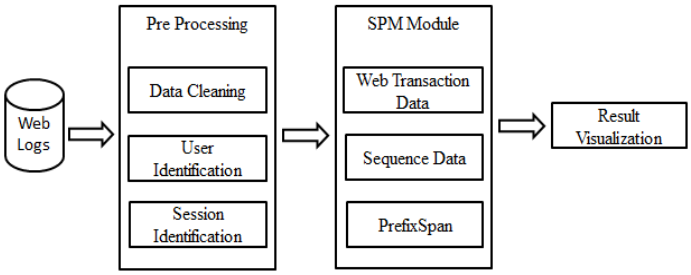

Figure 2: System Architecture

We analyze the efficiency of the algorithm as follows:

- No candidate sequence needs to be generated by PrefixSpan. Unlike a priori-like algorithms, Prefix-Span only grows longer sequential patterns from the shorter frequent ones. It neither generates nor tests any candidate sequence nonexistent in a projected database.
- Projected databases keep shrinking. A projected database is smaller than the original one because only the suffix subsequences of a frequent prefix are projected into a projected database. The shrinking factors can be significant because 1) usually, only a small set of sequential patterns grow quite long in a sequence database and, thus, the number of sequences in a projected database usually reduces substantially when prefix grows; and 2) projection only takes the whole string (not just suffix) and, thus, the shrinking factor is less than that of PrefixSpan.
- The major cost of PrefixSpan is the construction of projected databases. In the worst case, PrefixSpan constructs a projected database for every sequential pattern. If there exist a good number of sequential patterns, the cost is nontrivial. Techniques for reducing the number of projected databases will be discussed in the next subsection.

IV. EXPERIMENTAL RESULTS

To evaluate the effectiveness and efficiency of the PrefixSpan algorithm, we performed performance study of the algorithm by varying min_sup on data sets, with various kinds of sizes.

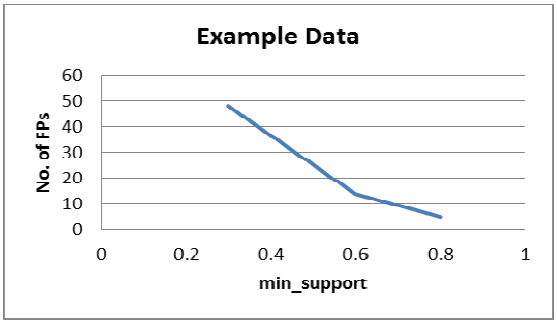

Figure 3: Distribution of Frequent Patterns of Example Data Set

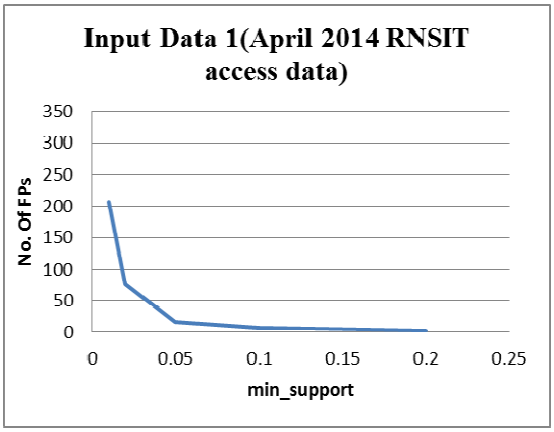

Figure 4: Distribution of Frequent Patterns for April 2014 RNSIT Access Data

CONCLUSIONS

We have performed a systematic study on mining of sequential patterns in large databases and developed a pattern-growth approach for efficient and scalable mining of sequential patterns.

Instead of refinement of the a priori-like, candidate generation-and-test approach, such as GSP, we promote a divide-and-conquer approach, called pattern-growth approach, which is an extension of FP-growth, an efficient pattern-growth algorithm for mining frequent patterns without candidate generation. Sequential pattern mining has broad applications including web reorganization, personalized marketing, customer retention.

ACKNOWLEDGEMENTS

The Author would like to thank the management of R N S Institute of Technology- HOD Dept. CSE, Dr G T Raju, Principal Dr. M K Venkatesha, and Director Dr. H N Shivashankar for the encouragement.

REFERENCES

1. R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” Proc. 1994 Int’l Conf. Very Large Data Bases(VLDB ’94), pp. 487-499, Sept. 1994.
2. R. Agrawal and R. Srikant, “Mining Sequential Patterns,” Proc.1995 Int’l Conf. Data Eng. (ICDE ’95), pp. 3-14, Mar. 1995.
3. R.J. Bayardo, “Efficiently Mining Long Patterns from Databases,”Proc. 1998 ACM-SIGMOD Int’l Conf. Management of Data (SIGMOD ’98), pp. 85-93, June 1998.
4. R.J. Bayardo, R. Agrawal, and D. Gunopulos, “Constraint-BasedRule Mining on Large, Dense Data Sets,” Proc. 1999 Int’l Conf. DataEng. (ICDE ’99), pp. 188-197, Apr. 1999.
5. C. Bettini, X.S. Wang, and S. Jajodia, “Mining Temporal Relationshipswith Multiple Granularities in Time Sequences,” Data Eng.Bull., vol. 21, pp. 32-38, 1998.
6. S. Guha, R. Rastogi, and K. Shim, “Rock: A Robust ClusteringAlgorithm for Categorical Attributes,”

Proc. 1999 Int'l Conf. DataEng. (ICDE '99), pp. 512-521, Mar. 1999.

7. J. Han, G. Dong, and Y. Yin, "Efficient Mining of Partial PeriodicPatterns in Time Series Database," Proc. 1999 Int'l Conf. Data Eng.(ICDE '99), pp. 106-115, Apr. 1999.
8. J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu,"FreeSpan: Frequent Pattern-Projected Sequential PatternMining," Proc. 2000 ACM SIGKDD Int'l Conf. Knowledge Discoveryin Databases (KDD '00), pp. 355-359, Aug. 2000.