MINING SEQUENTIAL PATTERNS WITH REGULAR EXPRESSION CONSTRAINTS USING SEQUENTIAL PATTERN TREE

Meer Hamza, Khaled Mahar, Mohamed Younis
Arab Academy for Science and Technology and Maritime Transport, Alexandria, Egypt

Key words: data mining, sequential patterns, regular expression, constraints

Abstract: The significant growth of sequence database sizes in recent years increase the importance of developing new techniques for data organization and query processing. Discovering sequential patterns is an important problem in data mining with a host of application domains. For effectiveness and efficiency consideration, constraints are essential for many sequential applications. In this paper, we give a brief review of different sequential pattern mining algorithms, and then introduce a new algorithm (termed NewSPIRIT) for mining frequent sequential patterns that satisfy user specified regular expression constraints. The general idea of our algorithm is to use a finite state automata to represent the regular expression constraints and build a sequential pattern tree that represents all sequences of data which satisfy this constraints by scanning the database of sequences only once. Experimental results shows that our NewSPIRIT is much more efficient than existing algorithms.

1 INTRODUCTION

Sequential pattern mining from a large database of sequences is an important data mining problem with broad applications. Briefly, given a set of data sequences, the problem is to discover subsequences that are *frequent*, in the sense that the percentage of data sequences containing them exceeds a user-specified minimum *support* (Agrawal,1995, Srikant, 1996). Mining frequent sequential patterns has found a host of potential application domains, including retailing (i.e., market-basket data), telecommunications, and, more recently, the World Wide Web (WWW).

There have been many studies on efficient sequential pattern mining algorithms and their applications (e.g. (Agrawal,1995, Srikant, 1996, Mannila, 1997, Pei, 2001, Zaki, 2001)). Sequential pattern mining algorithms, in general, can be categorized into three classes: (1) Apriori-based, with horizontal formatting method, with GSP (Srikant, 1996) as its representative; (2)Apriori-based, with vertical formatting method, such as SPADE (Zaki, 2001); and (3) projection-based pattern growth method, such as PrefixSpan (Pei, 2001).

For effectiveness and efficiency considerations, constraints are essential in many data mining applications. In the context of constraint-based sequential pattern mining, Srikant and Agrawal (Srikant, 1996) generalize the scope of sequential pattern mining (Agrawal,1995) to include time constraints, sliding time windows, and user-defined taxonomy. Garofalakis et al. (Garofalakis, 1999) proposed regular expressions as constraints for sequential pattern mining and develop a family of SPIRIT algorithms. In the *SPIRIT* framework, pattern constrains are specified as regular expressions, which is an especially convenient method if a user wants to significantly restrict the structure of patterns to be discovered. It has been shown that pushing regular expression constraints deep into the mining process can reduce processing time by more than an order of magnitude.

The remainder of this paper is organized as follow: section 2 gives a formal statement of sequential pattern mining problem, section 3 gives a formal statement of mining sequential pattern with constraint, while section 4 address the problem of sequential pattern mining with Regular Expression constrains explaining SPIRIT algorithms and its drawback. Finally, we will introduce a new efficient algorithm for this problem named “NewSPIRIT” in section 5 and its experimental results and performance analysis in section 6.

Hamza M., Mahar K. and Younis M. (2004).
MINING SEQUENTIAL PATTERNS WITH REGULAR EXPRESSION CONSTRAINTS USING SEQUENTIAL PATTERN TREE.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 116-121
DOI: 10.5220/0002621601160121

2 SEQUENTIAL PATTERN MINING PROBLEM

In this section, we first define the problem of sequential pattern mining, and then illustrate some of the well known Sequential Pattern Mining Algorithms explaining their working with examples.

2.1 Definitions

The formal statement of sequential pattern mining is defined in (Agrawal,1995) as following:
Let $I = \{x_1,\ldots,x_m\}$ be a set of **items**. An **itemset** is a non-empty subset of items, and an itemset with l items called a l-**itemset**.
A **sequence** $s = \langle s_1,\ldots,s_n \rangle$ is an ordered list of itemsets where s_i is the i^{th} element of s and is called a transaction. The number of transaction in a sequence is called the **length** of the sequence. A sequnce s with length k is called k-**sequence** and is denoted by $|s|$.
Consider two data sequences $s = \langle s_1,\ldots,s_n \rangle$ and $t = \langle t_1,\ldots,t_m \rangle$. We say that s is a **subsequence** of t if s is a "projection" of t derived by deleting elements and/or items from t. More formally s is a **subsequence** of t if there exist integers $j_1 < j_2 < j_3 < \ldots < j_n$ such that $s_1 \subseteq t_{j1}$, $s_2 \subseteq t_{j2},\ldots,$ *and* $s_n \subseteq t_{jn}$. For example sequences $\langle 1\ 3 \rangle$ and $\langle 1\ 2\ 4 \rangle$ are subsequences of $\langle 1\ 2\ 3\ 4 \rangle$, while $\langle 3\ 1 \rangle$ is not.
Following (Srikant, 1996), the sequence s is defined to *be subsequence with a maximum distance constraint of* δ, or alternately ***δ-distance subsequence***, of t if there exist integers $j_1 < j_2 < j_3 < \ldots < j_n$ such that $s_1 \subseteq t_{j1}$, $s_2 \subseteq t_{j2}$, $s_n \subseteq t_{jn}$ and $j_k - j_{k-1} \leq \delta$ for each $k = 2,3,4,\ldots,n$. That is, occurrences of adjacent elements of s within t are not separated by more than δ elements.
As a special case of the above definition, we say that s is a **contiguous subsequence** of t if s is a 1-distance subsequence of t, i.e., the elements of s can be mapped to a contiguous segment of t.
A sequence s is said to **contain** a sequence p if p is a subsequence of s.
The **support** of a pattern p is defined as the fraction of sequences in the input database that contain p.
Given a set of sequences S, we say that $s \in S$ is **maximal** if there are no sequences in $S - \{ s \}$ that contain it.

2.2 Sequential Pattern Mining Algorithms

Sequential pattern mining has been intensively studied during recent years, so there exist a great diversity of algorithms for sequential pattern mining. Most of these algorithms are based on the Apriori property proposed in association rule mining (Agrwal, 1994), which states that any sub-pattern of a frequent pattern must be frequent. Based on this heuristic, a series of Apriori-like algorithms have been proposed: AprioriAll, AprioriSome, DynamicSome in (Agrawal,1995), and GSP (Srikant, 1996). Later on another series of data projection based algorithms became popular because of their efficiency, which include FreeSpan (Han, 2000) and PrefixSpan (Pei, 2001). Recently, Zaki proposed an efficient algorithm called SPADE (Zaki, 2001), which is a lattice based algorithm. After that, a fast algorithm, called SPAM (Ayres, 2002) is proposed, it uses a vertical bitmap representation of the data. Also, a memory indexing based approach called MEMISP (Ming-Yen, 2002) is proposed, it uses a memory indexing scheme to reduce the I/O complexity.

3 SEQUENTIAL PATTERN MINING AND CONSTRAINTS

Like many frequent mining problems, there are two major difficulties in sequential pattern mining: (1) *effictiveness:* mining may return a huge number of patterns, many of which could be uninteresting to users, and (2) *efficiency*: it often takes substantial processng power for mining the complete set of sequential paterns in a large sequence database. Constraint-based mining may overcome both difficulties since constraints usually represents user's interest and focus, which confines the patterns to be found to a particular set o conditions. Moreover, if constraints can be pushed deep into the mining process, it is likely to achieve efficiency since the search can be focused. This motivates the study of constraint-based mining of sequential patterns.

3.1 Categories of Constraints

For real-world data mining, it is interesting to examine some interesting constraints from the application point of view. These constraints are presented in (Pei, 2002). Although this is by no means complete, it covers most of the interesting constraints in applications.
Alternatively, constraints can be categorized according to their properties for constraint pushing in the candidate generation and pruning processes(Ng, 1998, Pei, 2000, Pei, 2001). *Monotonicity*, *anti-monotonicity*, and *succinctness* are three categories of constraints that we briefly discuss below.

- A constraint C_A is anti-monotonic if a sequence *s* satisfying C_A implies that every nonempty subsequence of *s* also satisfies C_A.
- A constraint C_M is monotonic if a sequence *s* satisfies C_M implies that every super sequence of *s* also satisfies C_M.
- The basic idea behind succinct constraint is that, with such a constraint, one can explicitly and precisely generate all the sets of items satisfying the constraint without recourse to a generate-everything-and-test approach. A succinct constraints is specified using a precise "formula". According to the "formula", one can generate all the patterns satisfying a succinct constraint.

3.2 Mining Sequential Patterns with Constraints

The classical constraint-pushing framework based on anti-monotonicity, monotonicity, and succinctness can be applied to a large class of constraints (Ng, 1998). Thus the corresponding constraint-pushing strategy can be integrated easily into any one of sequential pattern mining algorithms, such as GSP, SPADE, and PrefixSpan. However, some important classes of constraints, such as RE (regular expressions), average, and sum, do not fit into this framework.

Sequential pattern can also be mined using level-by-level, candidate-generation-and-test Apriori-like methods. For example to find a sequential pattern ⟨*abc*⟩, which is an ordered list of three transactions {a}, {b}, and {c}, we can first find frequent length-1 patterns ⟨ *a* ⟩, ⟨ *b* ⟩, and ⟨ *c* ⟩. Then, length-2 candidates ⟨ *aa* ⟩, ⟨ *ab* ⟩, …., ⟨ (*ab*) ⟩, ⟨ (*ac*) ⟩, and ⟨(*bc*)⟩ can be generated and tested. If ⟨ *ab* ⟩, ⟨ *ac* ⟩, and ⟨ *bc* ⟩ are all frequent, length-3 candidate ⟨ *abc* ⟩ can be generated and tested. Even Apriori-like sequential pattern mining methods are intuitive extensions, they meet difficulties in handling many constraints (Pei, 2002).

4 SEQUENTIAL PATTERN MINING WITH REGULAR EXPRESSION CONSTRAINT

A **regular expression** (RE) constraint *R* is specified as a RE over the alphabet of sequence elements using the established set of RE operators, such as disjunction (|) and Kleene closure (*) (Garofalakis, 1999). Thus, a RE constraint *R* specifies a language of strings over the element alphabet or, equivalently, a regular family of sequential patterns that is of interest to the user. A well-known result from complexity theory states that REs have exactly the same expressive power as *deterministic finite automata* (Garofalakis, 1999). Thus, given any RE *R*, we can always build a deterministic finite automaton A_R such that A_R accepts exactly the language generated by *R*. Figure 2 depicts the state diagram of a deterministic finite automaton for the RE 1*(22|234|44) (i.e., all sequences of zero or more 1′s followed by 2 2, 2 3 4, or 44) Following (Garofalakis, 1999), we use double circles to indicate an accept state and > to emphasize the start state (a) of the automaton.

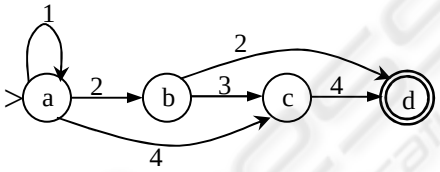


Figure 2: Automaton for the RE 1*(22|234|44)

4.1 Problem Statement

The abstract definition of our constrained pattern mining problem is as follows.

- **Given:** A database of sequences *D*, a user-specified minimum support threshold, and a user-specified RE constraint *R* (or, equivalently, an automaton A_R).
- **Find:** All *frequent and valid* sequential patterns in *D*.

Thus, the objective is to efficiently mine patterns that are not only frequent but also belong to the language of sequences generated by the RE *R*.

4.2 The SPIRIT Algorithms

Garofalakis et al. proposed a family of algorithms that uses the RE as a flexible constraint specification tool that enable users to focus on what he needs from the mining process (Garofalakis, 1999). Using an input parameter *C* to denote a generic user-specified constraint on the mined patterns. The output of a SPIRIT algorithm is the set of frequent sequences in the database *D* that satisfy constraint *C*. The algorithmic framework is similar in structure to the general Apriori strategy of Agrawal and Srikant (Srikant, 1996). The candidate counting step is typically the most expensive step of pattern mining process and its overhead is directly proportional to the size of C_k, Thus the goal of an efficient pattern mining strategy is to employ minimum support requirements and any additional user-specified constraint to restrict as much as possible the set of candidate *k*-sequences counted during the pass k.

The SPIRIT framework strives to achieve this goal by using two different types of pruning within each pass *k*.

- *Constraint-based pruning* using a relaxation C' of the user-specified constraint C; that is, ensuring that all candidate *k*-sequences in C_k satisfy C'. This is accomplished by appropriately employing C' and F in the candidate generation phase.
- *Support-based pruning*; that is, ensuring that all subsequences of a sequence s in C_k that satisfy C' are present in the current set of discovered frequent sequences F.

Intuitively, constraint-based pruning tries to restrict C_k by (partially) enforcing the input constraint C, where as support-based pruning tries to restrict C_k by checking the minimum support constraint for qualifying subsequences. Note that, given a set of candidates C_k and a relaxation C' of C, the amount of support-based pruning is maximized when C' is *anti-monotone* (i.e. all subsequences of sequence satisfying C' are guaranteed also to satisfy C'). This is because support information for *all* of the subsequences of a candidate sequence s in C_k can be used to prune it. However, when C' is *not* anti-monotone, the amounts of constraint-based and support-based pruning achieved vary depending on the specific choice of C'.

5 NEWSPIRIT: A NEW ALGORITHM FOR SEQUENTIAL PATTERN MINING WITH REGULAR EXPRESSION CONSTRAINTS

In this section, the proposed algorithm for sequential pattern mining with constraint named, NewSPIRIT is described. NewSPIRIT scans the sequence database only once to build the sequence-pattern tree in the main memory and get the valid frequent sequences by traversing this tree using a standard depth-first search (DFS) manner.

5.1 Sequential Pattern Tree Design and Construction

The algorithm represents the discovered patterns in a tree structure called *SP-Tree* (sequential-pattern tree). This tree is growing progressively by the algorithm such that each valid sequence is represented in the tree with its support count. We assumes that the *SP-Tree* (and all data structures used for the algorithm) completely fit into main memory, this is because the size of current main memories reaching gigabytes and still growing.

Each sequence in the *SP-Tree* can be considered as either a sequence-extended sequence or an itemset-extended sequence. A sequence-extended sequence is a sequence generated by adding a new transaction consisting of a single item to the end of its parent's sequence in the tree. An itemset-extended sequence is a sequence generated by adding an item to the last itemset in the parent's sequence, such that the item is greater than any item in that last itemset. For example, if we have a sequence s_a = ⟨ (a b) a ⟩, then ⟨ (a b) a c ⟩ is an extended sequences of s_a but ⟨ (a b) (a c) ⟩ is an itemset-extended sequence of s_a. In our algorithm we extend this definition by using the interval time between adjacent elements in a pattern as a dimension for counting the support for each pattern. Fig. 3 shows a sample of *SP-Tree*, where each node has its sequence extension, the interval time between it and its parent , and the support count for this interval. Note that, itemset extension occurs when the interval time = 0, otherwise the extension will be sequence extension.

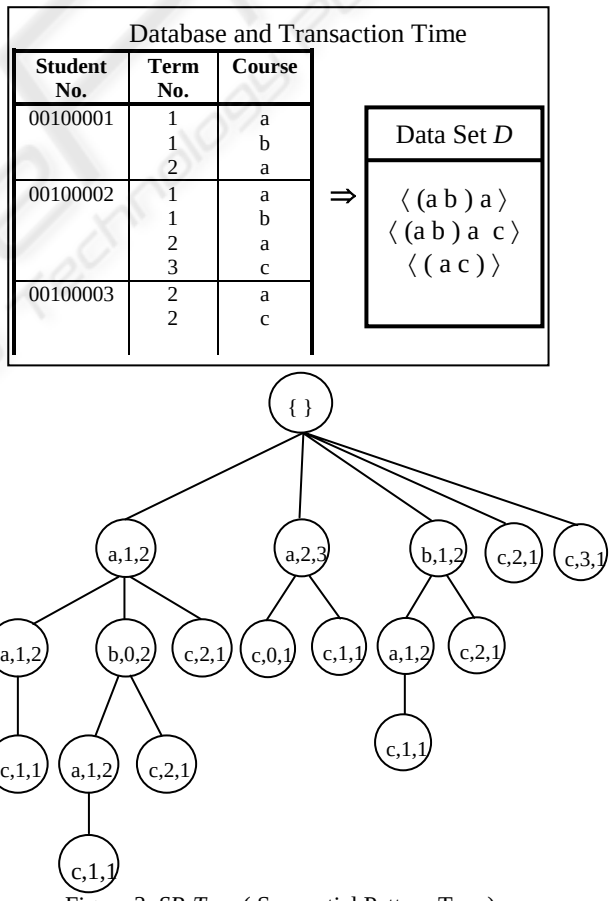


Figure 3. *SP-Tree* (Sequential Pattern Tree)

5.2 The NewSPIRIT algorithm

Fig. 4 depicts the algorithmic skeleton of the NewSPIRIT algorithm, using an input constraint *C* to denote a generic user-specified constraint on the mined patterns. The output of the NewSPIRIT is the set of frequent sequences in the database *D* that satisfy constraint *C*. The NewSPIRIT needs only one database scan to construct the *SP-Tree* which its sequences are valid with respect to the constraint *C*. Note that, the *SP-Tree* is built using constraint based pruning only, where the support count is become known after finishing the sequence tree building. After that the support based pruning (that is, ensuring that all sequences are frequent) is achieved by traversing *SP-Tree* in DFS manner.

Algorithm : NewSPIRIT Mine sequential patterns with regular expression constraint using *SP-Tree.*
Input : A transaction Database, *D*; minimum support threshold, min_sup; Regular expression Constraint *C.*
Output : The complete set of sequential patterns *F*
Method :-
- Read the regular expression constraint *C* and construct the finite state automata
- Create the root of the *SP-Tree* and label it as "Null"
- Foreach sequence *s* in the database *D*
  - Foreach valid subsequence *t* from *s*
    - Parse the *SP-Tree* with the valid sequence *t* , visit its nodes, and then insert the remainder part of *t* that is not found in it.
  - End foreach
- End Foreach
- Traverse the SP-Tree to get the complete set of sequential patterns where each path on the SP-Tree is a sequence with support count at the last node on this path.

Figure 4: NewSPIRIT Algorithm

5.3 *SP-Tree* Traversal

The NewSIRIT traverses the sequence tree described above in a standard DFS manner to get all valid sequences with its support count. The support count is the minimum count from the nodes that represent this sequence (i.e. from the last sequence extension for this sequence). At each node n, the support of each sequence-extended child is tested. If the support of a generated sequence *s* is greater than or equal to minSup, we store that sequence and repeat DFS recursively on *s*. If the support of *s* is less than minSup, then we do not need to repeat DFS on *s* by the Apriori principle, since any child sequence generated from *s* will not be frequent (Srikant, 1996). If none of the generated children are frequent, then the node is a leaf and we can backtrack up the tree.

6 EXPERIMENTAL RESULTS

To evaluate the effectiveness and efficiency of the NewSPIRIT algorithm, we performed an extensive study of four algorithms: SPIRIT(L), SPIRIT(V), SPIRIT(R) and NewSPIRIT on many samples of data sets, with various kinds of sizes and data distributions. The implementation of these algorithms was done to compare between them at the same platform, and at the same environment. Detailed algorithm implementation is described as follows.

- SPIRIT(L), SPIRIT(V), and SPIRIT(R) algorithms are implemented according to the description in (Garofalakis, 1999).
- NewSPIRIT is implemented as described in this thesis .

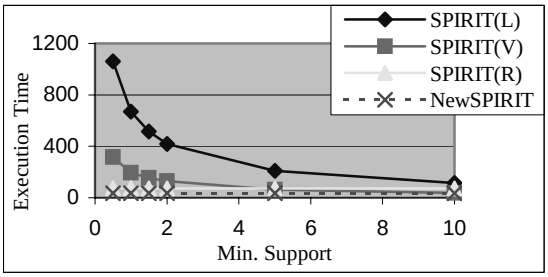
For the data sets used in our performance study, we use two samples of data sets from real database. We have obtained these samples of data sets from student database in AAST. The sequences are generated from the student course registration table which contain student-id, counse-id, and registration term. The Regular Expression constraint are constructed from the course plan of a specified department. In these data sets the number of items were 70 item (Number of courses in the course plan in this department), while the number of sequences and its average lengths are :-

- 1323 sequence with average length 47 for data set I
- 253 sequence with average length 17 for data set II

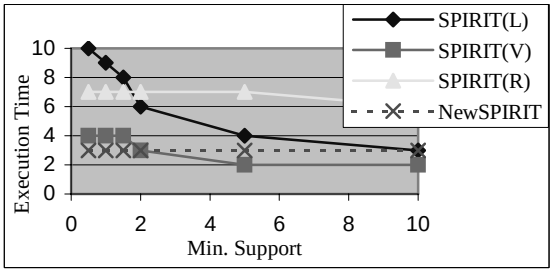
The testing result in figure 5 makes clear distinction among the algorithms tested, where it shows that our NewSPIRIT is much more efficient than SPIRIT(V), SPIRIT(R), and SPIRIT(L). The execution time of our NewSPIRIT is independent of min. support threshold where it constructs its SP-Tree based only on the Regular Expression constraint. With the above comprehensive performance study, we are convinced that NewSPIRIT is the clear winner among all the four tested algorithms. The other three algorithms (SPIRIT(L), SPIRIT(V), and SPIRIT(R) require many passes over the databases where they are based on GSP. GSP method that it bear three nontrivial, inherent costs which are independent of detailed implementation techniques.

1- A huge set of candidate sequences could be generated in a large sequence database.
2- Many scans of databases in mining.

3- The Aperiori-based method encounters difficulty when mining long sequential patterns.



a. Performance results for 1323 sequence with 47 average length



b. Performance results for 253 sequence with 17 average length

Figure 5: Experimental Results

Our NewSPIRIT require only one pass over the database, no candidate generation. It needs only a sufficient memory for SP-Tree which it is restricted by the regular expression constrant.

Figure 6 shows the results of scalability test of the four algorithms on data set II. The database is growing to multiples of data set II, with min. support = 1 % . As shown in the figure, the times for all algorithms linearly scale with data set size. This is because the number of candidates generated by each algorithm is independent of the data set size.

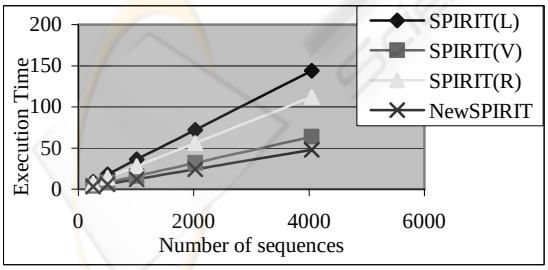


Figure 6: Scalability test of the four algorithms on data set II, with min. support 1%

7 CONCLUSION

In this paper we introduce the problem of mining sequential patterns with constraints, and presented a new algorithm for sequential pattern mining with regular expression constraint (NewSPIRIT) that can efficiently find all frequent sequential patterns that satisfy a regular expression constraint. NewSPIRIT mines the set of all sequential patterns without generating candidates or sub-databases and achieve our goal by scanning the database only once.

REFERENCES

Agrawal, R. and Srikant, R., 1995. Mining Sequential Patterns. In *Proc. of the 11th Intl. Conf. on Data Engineering*.

Srikant, R. and Agrawal, R., 1996. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proc. of the 5th Intl. Conf. on Extending Database Technology (EDBT'96)*.

Mannila, H., Toivonen, H. and Verkamo, A. I., 1997. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289.

Pei, J. et al, 2001. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. *CDE'01*.

Zaki, M., 2001. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 40:31–60.

Garofalakis, M., Rastogi, R. and Shim, K., 1999. Spirit: Sequential pattern mining with regular expression constraints. *VLDB'99*.

Agrwal, R. and Srikant, R., 1994. Fast algoritms for mining association rules. *VLDB'94*.

Han, J., Pei, J., Mortazavi-As1, B., Chen, Q., Dayal, U. and M-C. Hsu, 2000. FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining. *In Proc. 2000 Int. Conf. Knoweldge Discovery and Data Mining (KDD'00)*, 355 –359, Boston, MA.

Ayres, J., Gehrke, J., Yiu, T. and Flannick, J., 2002. Sequential pattern Mining using A Bitmap Representation. *SIGKDD'02*.

Ming-Yen Lin and Suh-Yin Lee, 2002. Fast discovery of sequential patterns by memory indexing. In *Proc. of 2002 DaWaK*, pages 150-160.

Pei, J., Han, J. and Wei Wang, 2002. Mining Sequential patterns with constraints in Large Databases. *CIKM'02*.

Pei, J. and Han, J., 2002. Constraints Frequent Pattern Mining: A Pattern-Growth View. *ACM SIGKDD Explorations (Special Issue on Constraints in Data Mining)*, Volume 4, Issue 1, pages 31-39.

Ng, R., Lakshmanan, L. V. S., Han, J., and A. Pang, 1998. Exploratory mining and pruning optimizations of constrained associations rules. *SIGMOD'98*.

Pei, J. and Han, J., 2000. Can we push more constraints into frequent pattern mining? *KDD'00*.

Pei, J. et al., 2001. Mining frequent itemsets with convertible constraints. *ICDE'01*.