FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining *

Jiawei Han, Jian Pei, Behzad Mortazavi-Asl
Intelligent Database Systems Research Lab.
School of Computing Science
Simon Fraser University
Burnaby, B.C., Canada V5A 1S6
E-mail: {han, peijian, mortazav}@cs.sfu.ca

Qiming Chen, Umeshwar Dayal, Mei-Chun Hsu
Hewlett-Packard Labs.
1501 Page Mills Road, P.O. Box 10490
Palo Alto, California 94303-0969
U.S.A.
E-mail: {qchen, dayal, mchsu}@hpl.hp.com

Abstract

Sequential pattern mining is an important data mining problem with broad applications. It is also a difficult problem since one may need to examine a combinatorially explosive number of possible subsequence patterns. Most of the previously developed sequential pattern mining methods follow the methodology of Apriori since the Apriori-based method may substantially reduce the number of combinations to be examined. However, Apriori still encounters problems when a sequence database is large and/or when sequential patterns to be mined are numerous and/or long.

In this paper, we re-examine the sequential pattern mining problem and propose a novel, efficient sequential pattern mining method, called FreeSpan (i.e., **Fre**quent pattern-projected **S**equential **pattern** mining). The general idea of the method is to integrate the mining of frequent sequences with that of frequent patterns and use projected sequence databases to confine the search and the growth of subsequence fragments. FreeSpan mines the complete set of patterns but greatly reduces the efforts of candidate subsequence generation. Our performance study shows that FreeSpan examines a substantially smaller number of combinations of subsequences and runs considerably faster than the Apriori-based GSP algorithm.

1 Introduction

Sequential pattern mining, which discovers frequent subsequences as patterns in a sequence database, is an important data mining research problem with broad applications. Since its first introduction in [3], many studies have contributed to the efficient mining of sequential patterns or other frequent patterns in time-related data [3, 6, 5]. Almost all the methods proposed so far for mining sequential patterns and other time-related frequent patterns are Apriori-like, i.e., based on the Apriori heuristic first proposed in association mining [2]: *any super-pattern of a nonfrequent pattern cannot be frequent.* Based on this heuristic, a typical Apriori-like method such as GSP [6] adopts a multiple-pass, candidate generation-and-test approach in sequential pattern mining.

The Apriori-like sequential pattern mining methods, though they reduce the search space, bear three nontrivial, inherent costs which 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, if there are 1000 frequent sequences of length-1, an Apriori-like algorithm will generate $1000 \times 1000 + \frac{1000 \times 999}{2} = 1,499,500$ candidate sequences.
- **Many scans of databases in mining.** Since the length of each candidate sequence grows by one at each database scan, to find a sequential pattern $\{(abc)\ (abc)(abc)(abc)(abc)\}$, the Apriori-based method must scan the database at least 15 times. This bears a nontrivial cost.
- **The Apriori-based method encounters difficulty when mining long sequential patterns.** This is because a long sequential pattern must grow up from a huge number of short sequential patterns, but the number of such candidate sequences is exponential to the length of the sequential patterns to be mined.

Based on our analysis, both the thrust and the bottleneck of an Apriori-based sequential pattern mining method come from its step-wise candidate sequence generation and test. Can we develop a new method which may absorb the spirit of Apriori but avoid or substantially reduce the expensive candidate generation and test? Recent researches [4,

* The work was supported in part by the Natural Sciences and Engineering Research Council of Canada (grant NSERC-A3723), the Networks of Centres of Excellence of Canada (grant NCE/IRIS-3), and the Hewlett-Packard Lab, U.S.A.

1] explore the techniques for projection databases to achieve high performance in frequent pattern mining. Can projected databases facilitate sequential pattern mining? This is the motivation of this study.

In this paper, we develop a new sequential mining method, called FreeSpan (i.e., **Fre**qu**e**nt pattern-projected **S**equential **pa**ttern mining). Its general idea is to use frequent items to recursively project sequence databases into a set of smaller projected databases and grow subsequence fragments in each projected database. This process partitions both the data and the set of frequent patterns to be tested, and confines each test being conducted to the corresponding smaller projected database. Our experimental performance studies show that FreeSpan mines the complete set of patterns and is efficient and runs considerably faster than the Apriori-based GSP algorithm.

The remaining of the paper is organized as follows. In Section 2, we present our new method FreeSpan. The experimental and performance results are presented in Section 3. We summarize our study in Section 4.

2 FreeSpan

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of **items**. An **itemset** is a subset of items. A **sequence** is an ordered list of itemsets. A sequence s is denoted by $\langle s_1 s_2 \cdots s_l \rangle$, where s_j is an itemset, i.e., $s_j \subseteq I$ for $1 \leq j \leq l$. s_j is also called an **element** of the sequence, and denoted as $(x_1 x_2 \cdots x_m)$, where x_k is an item, i.e., $x_k \in I$ for $1 \leq k \leq m$. For brevity, the brackets are omitted if an element has only one item. That is, 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. A sequence $\alpha = \langle a_1 a_2 \cdots a_n \rangle$ is called a **subsequence** of another sequence $\beta = \langle b_1 b_2 \cdots b_m \rangle$ and β a **super sequence** of α, denoted as $\alpha \sqsubseteq \beta$, if there exist integers $1 \leq j_1 < j_2 < \cdots < j_n \leq m$ such that $a_1 \subseteq b_{j_1}$, $a_2 \subseteq b_{j_2}$, $\ldots$, $a_n \subseteq b_{j_n}$.

A **sequence database** S is a set of tuples $\langle sid, s \rangle$, where sid is a **sequence_id** and s is a sequence. A tuple $\langle sid, s \rangle$ in a sequence database S is said to *contain* a sequence α, if α is a subsequence of s, i.e., $\alpha \sqsubseteq s$. The **support** of a sequence α in a sequence database S is the number of tuples in the database containing α. Given a positive integer ξ as the **support threshold**, a sequence α is called a **sequential pattern** in sequence database S if the sequence is contained by at least ξ tuples in the database.

Now, we introduce our novel mining method, FreeSpan, using the following example.

Given the sequence database S in the first two columns in Table 1 and $min\_support = 2$. Let us mine sequential patterns.

By scanning S once, we find the set of frequent items, L_1, which is also the set of length-1 sequential patterns. These frequent items are sorted in *support descending order*, i.e. $\langle b:5, c:4, a:3, d:3, e:3, f:3 \rangle$. It is called **frequent item list** or simply f_list. It will be used throughout the mining process. The complete set of sequential patterns can be divided into six subsets without overlap: those having item f, those having item e but no f, those having item d but no e nor f, and so on. FreeSpan uses such a divide-and-conquer method to find the complete set of sequential patterns.

Table 1: A sequence database

Sequence_id	Sequence	item pattern
10	$\langle (bd)cb(ac) \rangle$	$\{a, b, c, d\}$
20	$\langle (bf)(ce)b(fg) \rangle$	$\{b, c, e, f, g\}$
30	$\langle (ah)(bf)abf \rangle$	$\{a, b, f, h\}$
40	$\langle (be)(ce)d \rangle$	$\{b, c, d, e\}$
50	$\langle a(bd)bcb(ade) \rangle$	$\{a, b, c, d, e\}$

Then, we construct a **frequent item matrix** F to count the occurrence frequency of each length-2 sequence formed by items in the f_list, as follows. For f_list $\langle i_1, i_2, \ldots, i_n \rangle$, F is a triangular matrix $F[j,k]$, where $1 \leq j \leq m$ and $1 \leq k \leq j$. $F[j,j]$ (for $1 \leq j \leq m$) has only one counter, whereas every other slot $F[j,k]$ $(1 < j \leq m$, and $1 \leq k < j)$ has three counters: (A, B, C), where A is the number of occurrences that i_k occurs *after* i_j (i.e., the sequence contains $\langle i_j i_k \rangle$), B is that i_k occurs *before* i_j (i.e., the sequence contains $\langle i_k i_j \rangle$), and C is that i_k occurs *concurrently with* i_j (i.e., the sequence contains $\langle (i_j i_k) \rangle$).

In this example, the f_list contains six items. It leads to the generation of a 6×6 triangular frequent item matrix, with every counter initialized to 0. The second scan fills up the matrix as follows. The first sequence $\langle (bd)cb(ac) \rangle$ increases the first two counters of matrix $F[b,c]$ by 1, i.e., $F[b,c] = (1,1,0)$, since two cases, $\langle bc \rangle$ and $\langle cb \rangle$, but not $\langle (bc) \rangle$, occur here. For $F[b,d]$, since only $\langle (bd) \rangle$ and $\langle db \rangle$ occur in this sequence, we have $F[b,d] = (0,1,1)$. The process continues. Table 2 shows the resulting matrix after a complete scan.

Table 2: The frequent item matrix after the scan of S

b	4					
c	(4,3,0)	1				
a	(3,2,0)	(2,1,1)	2			
d	(2,2,2)	(2,2,0)	(1,2,1)	1		
e	(3,1,1)	(1,1,2)	(1,0,1)	(1,1,1)	1	
f	(2,2,2)	(1,1,0)	(1,1,0)	(0,0,0)	(1,1,0)	2
	b	c	a	d	e	f

This frequent item matrix is used to generate the length-2 sequential patterns and a set of projected databases, which are then used to generate length-3 and longer sequential patterns. For an itemset X, the **X-projected database** is the collection of sequences having all items in X. Infrequent items as well as items after those in X in f_list are discarded. Similarly, for a sequential pattern α, the **α-projected**

Table 3: Pattern generation from the frequent item matrix

Item	Output length-2 sequential patterns	Ann. on repeating items	Ann. on Projected DBs
f	$\langle bf\rangle : 2$, $\langle fb\rangle : 2$, $\langle (bf)\rangle : 2$,	$\{b^+f^+\}$	$\emptyset$
e	$\langle be\rangle : 3$, $\langle (ce)\rangle : 2$	$\langle b^+e\rangle$	$\langle (ce)\rangle : \{b\}$
d	$\langle bd\rangle : 2$, $\langle db\rangle : 2$, $\langle (bd)\rangle : 2$, $\langle cd\rangle : 2$, $\langle dc\rangle : 2$, $\langle da\rangle : 2$	$\{b^+d\}$, $\langle da^+\rangle$	$\langle da\rangle : \{b, c\}$, $\{cd\} : \{b\}$
a	$\langle ba\rangle : 3$, $\langle ab\rangle : 2$, $\langle ca\rangle : 2$, $\langle aa\rangle : 2$	$\langle aa^+\rangle$, $\{a^+b^+\}$, $\langle ca^+\rangle$	$\langle ca\rangle : \{b\}$
c	$\langle bc\rangle : 4$, $\langle cb\rangle : 3$,	$\{b^+c\}$	$\emptyset$
b	$\langle bb\rangle : 4$	$\langle bb^+\rangle$	$\emptyset$

database is the collection of sequences having α as a subsequence. Infrequent items and items after those in α are ignored.

In order to generate level-3 projected databases, a set of annotations, indicating which set of items or sequences should be examined in the projection and later mining of level-3 databases, should be the task of the next step.

Now, let us examine how to use the matrix to generate (1) length-2 sequential patterns, (2) annotations of item-repeating patterns, and (3) annotations of projected databases. The annotation of an item-repeating pattern is of the form $\$a_i^\gamma a_j^\gamma\$$, where $\$\ldots\$$ can be either $\langle\ldots\rangle$ (indicating looking for a particular ordered sequence only) or $\{\ldots\}$ (indicating looking for any ordered sequence), and a_i^γ can be either a_i^+ (indicating looking for more than one occurrence of a_i), or a_i (i.e., no repeating a_i's)—notice that at least one of a_i and a_j is a repeating one. Similarly, the annotation of the projected database is of the form $\$a_ia_j\$: \{b_p, \ldots, b_q\}$, where $\$\ldots\$$ has the same convention as the item repeating pattern, and $\{b_p, \ldots, b_q\}$ represents a set of frequent items which may occur together with $\$a_ia_j\$$ to form longer sequential patterns in subsequent mining.

More concretely, this step is performed as follows.

- generate length-2 sequential patterns: For each counter, if the value in the counter is no less than min_support, output the corresponding frequent pattern.
- generate annotations on item-repeating patterns for row j: For the diagonal slot, if $F[j,j] \geq min\_support$, generate $\langle jj^+\rangle$[1].

  For a column $i \neq j$, we have the following rules: if $F[i,i] \geq min\_support$, the annotation should contain i^+ [2]; if $F[j,j] \geq min\_support$, the annotation should contain j^+; and if only one of the three counters of $F[i,j]$ is frequent, *sequence* is used as the annotation; otherwise *set* is used[3].
- generate annotations on projected databases for row j: For each $i < j$, if $F[i,j]$, $F[k,j]$ and $F[i,k]$ ($k < i$) may form a pattern generating triple (i.e., all the corresponding pairs are frequent)[4], k should be added to i's projected column set. After examining all columns in front of i, the set of projected columns is determined. Output the annotation containing i, j and the set of projected columns. If there is a choice between *sequence* or *set*, sequence is preferred since it enforces a stronger restriction in the projection.

Length-2 sequential patterns and annotations on item-repeating patterns, and that on the projected databases are generated, as shown in Table 3, based on the frequent item matrix F (Table 2), We only show the process for two rows, f and e, here.

The f-row has $F[b,f] = (2,2,2)$, which generates three length-2 sequences: $\langle bf\rangle : 2$, $\langle fb\rangle : 2$, and $\langle (bf)\rangle : 2$. Since both $F[b,b]$ and $F[f,f]$ are frequent, the annotation $\{b^+f^+\}$ is generated which means one need to examine multiple occurrences of b's and f's and their combinations in the next scan. Since no other item could be co-frequent with $\langle bf\rangle$, there is no projected database annotation with f.

The e-row has two counters frequent, which leads to two length-2 sequences: $\langle be\rangle : 3$, and $\langle (ce)\rangle : 2$. Since $F[b,b]$ is frequent, we have the annotation: $\langle b^+e\rangle$. Moreover, since $F[c,e]$, $F[b,e]$, and $F[b,c]$ form a pattern generating triple, and $F[c,e] = (1,1,2)$ (which means only $\langle (ce)\rangle$ is valid), the annotation for the projected database should be $\langle (ce)\rangle : \{b\}$[5] which indicates generating $\langle (ce)\rangle$-projected database, with $\{b\}$ as the only additional items included.

After generating the annotations, the matrix can

[1] This indicates that the count of encountered jjj, $jjjj$, etc. should be registered in the next round scan to find whether they are frequent since jj has appeared more than min_support times.

[2] This means that there are potentially more than one i appearing in the sequential pattern.

[3] This distinction is used to enhance string filtering: annotation $\langle bf\rangle$ indicates there is no chance for the subsequence $\langle fb\rangle$ to survive but no so for $\{bf\}$.

[4] There is some subtlety to observe. For example, for min_support = 2, the tripe $F[e,f] = (1,9,0)$, $F[a,f] = (3,0,1)$, $F[a,e] = (1,1,3)$, is not a pattern generating triple because there is no way to satisfy simultaneously the three requirements $\langle fe\rangle$, $\langle af\rangle$ and $\langle (ae)\rangle$. However, if $F[a,e] = (3,1,1)$, it can generate a valid pattern $\langle afe\rangle$. For lack of space, the set of rules to generate the precise sequences or sets for all the combinations are omitted here.

[5] A more refined header should be $\langle (ce)\rangle : \{b\}[\langle be\rangle]$, which indicates that the next scan needs to include only the sequences containing ordered subsequence $\langle be\rangle$ besides (ce). We ignore this additional piece of information here since it costs an additional string match test, and its performance gain is not so obvious in many cases based on our experiments.

be discarded. That is, only the annotations are used in the third (and the last) scan of the database.

Based on the annotation generated from the matrix, we scan the database one more time and generate the item-repeating patterns and projected databases. The remaining mining will be confined to each small projected database, by examining only the corresponding patterns enclosed in the header set. Based on the annotations for item-repeating patterns and projected databases, we scan S one more time. The set item-repeating patterns generated is $\{\langle bbf\rangle$:2, $\langle fbf\rangle$:2, $\langle(bf)b\rangle$:2, $\langle(bf)f\rangle$:2, $\langle(bf)bf\rangle$:2, $\langle(bd)b\rangle$:2, $\langle bba\rangle$:2, $\langle aba\rangle$:2, $\langle abb\rangle$:2, $\langle bcb\rangle$:3, $\langle bbc\rangle$:2$\}$.

There are four projected databases: $\langle(ce)\rangle$:$\{b\}$, $\langle da\rangle$:$\{b, c\}$, $\{cd\}$:$\{b\}$ and $\langle ca\rangle$:$\{b\}$, as shown in Table 4. For a projected database whose annotation contains exactly three items, its associated sequential patterns can be obtained by a simple scan of the projected database. However, for a projected database whose annotation contains more than three items, one can construct frequent item matrix for this projected database and recursively mine its sequential patterns by the alternative-level projection technique.

We outline the FreeSpan algorithm as follows.

Algorithm 1 (FreeSpan) Given a sequence database S and the support threshold ξ, FreeSpan mine the complete set of sequential patterns as follows.

1. Scan S, find the set of frequent items in S, and (in frequency descending order) sort them into f_list.
2. Perform alternative-level projection mining which consists of the following steps: (1) construct a frequent item matrix by scanning the database once, (2) generate length-2 sequential patterns and the annotations on item-repeating patterns and projected databases, (3) scan database to generate item-repeating patterns and projected databases, and (4) do matrix projection mining on projected databases recursively, if there are still longer candidate patterns to be mined. □

3 Performance Study

In this section, we report our experimental results on the performance analysis of FreeSpan and its level-by-level projection version FreeSpan-1, in comparison with GSP , on scalability and processing efficiency. It shows that FreeSpan outperforms other previously proposed methods and is efficient and scalable for mining sequential patterns in large databases. All the experiments are performed on a 233MHz Pentium PC machine with 128 megabytes main memory. The synthetic datasets which we used for our experiments were generated using standard procedure described in [7]. In all the data sets, the number of items are set to $10,000$.

We compare performance of three methods. (1) GSP described in [7]; (2) FreeSpan-1 is FreeSpan but with level-by-level projection; and (3) FreeSpan which is a complete implementation of the method proposed in this paper.

The experimental results are shown in Figures 1. Graphs in Figure 1 (a) to (e) show the scalability of the methods as the support threshold decreases from 1% to 0.1%. For all the experiments using different sized sequence databases, with a different number of items in each element, and a different number of elements in each sequence, FreeSpan is always the clear winner. Naive FreeSpan and GSP are close, but lag increasingly behind when the data set grows large and the minimum support threshold reduces. You may also notice a sharp jump in the runtime of FreeSpan-1 when the support threshold falls below a certain point, because FreeSpan-1 requires more memory and will cause page thrashing.

We also tested the scalability of GSP and FreeSpan with the number of sequences in databases, with the results shown in the graph of Figure 1 (f). It shows both algorithms are linearly scalable, but FreeSpan has much better scalability.

Finally, we analyze the total space needed for FreeSpan. Since there are only three scans of the original database in FreeSpan, independent of the maximal length of the sequence, there is no need to have the sequence database residing in the main memory. One data page allocation should be good enough for the sequence database. Similarly, the bulky output of the computed frequent patterns can be swapped back to database as well. Thus the major memory space needed is to hold the frequent item matrix F. After this, the computation of projected database can be done one by one, and each will require much less memory than the original database.

For a database with m frequent distinct items, the total number of counters for frequent item matrix is $|F| = m + \frac{3}{2} \times (m-1) \times (m-2)$. If $m = 1000$ (i.e., 1000 frequent distinct items), $|F| \approx 1.5 \times 10^6 = 3$ megabytes (assuming each counter takes two bytes of space). This can be easily handled by today's computer. This space requirement will increase quadratically, e.g. computing the frequent item matrix for 10,000 frequent distinct items requires 300 megabytes memory space to hold the array. However, some memory management scheme (such as hash tree, etc.) can be used to ensure that the performance is not degraded seriously.

4 Conclusions

We have developed an interesting, scalable and efficient sequential pattern mining method, called FreeSpan, which integrates the mining of frequent sequences with that of frequent patterns and uses projected sequence databases to confine the search and the growth of subsequence fragments.

Our performance study shows that FreeSpan examines substantially fewer combinations of subsequences and runs considerably faster than the Apriori-based GSP algorithm. According to our analysis, the reasons that FreeSpan outperforms GSP are obvious: (1) FreeSpan projects a large sequence database recursively into a set of small projected sequence databases

Table 4: Four projected databases and their sequential patterns

annotation	$\langle(ce)\rangle$:{b}	$\langle da\rangle$:{b,c}	{cd}:{b}	$\langle ca\rangle$:{b}
projected database	$\langle b(ce)b\rangle$, $\langle b(ce)\rangle$	$\langle(bd)cb(ac)\rangle$, $\langle(bd)bcba\rangle$	$\langle(bd)cbc\rangle$, $\langle bcd\rangle$, $\langle(bd)bcbd\rangle$	$\langle bcba\rangle$, $\langle bbcba\rangle$
sequential patterns	$\langle b(ce)\rangle$:2	$\langle(bd)a\rangle$:2, $\langle dca\rangle$:2, $\langle dba\rangle$:2, $\langle(bd)ca\rangle$:2, $\langle(bd)ba\rangle$:2, $\langle dcba\rangle$:2, $\langle(bd)cba\rangle$:2	$\langle bcd\rangle$:2, $\langle(bd)c\rangle$:2, $\langle dcb\rangle$:2, $\langle(bd)cb\rangle$:2, $\langle(bd)bc\rangle$:2	$\langle bca\rangle$:2, $\langle cba\rangle$:2, $\langle bcba\rangle$:2

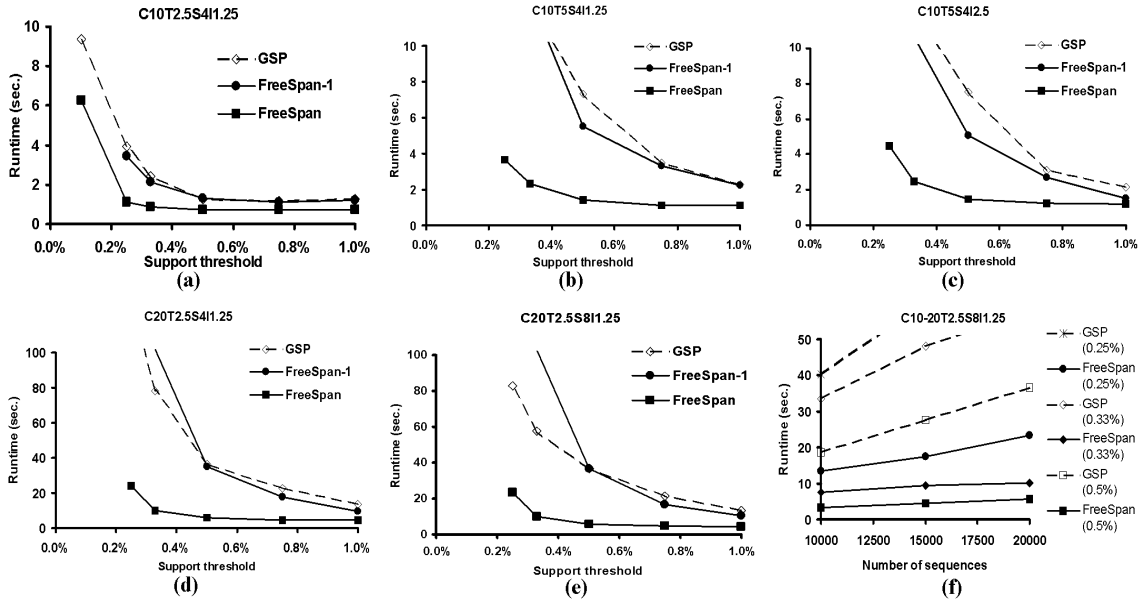

Figure 1: Performance Evaluation of Four Algorithms: GSP, FreeSpan-1, FreeSpan

based on the currently mined frequent sets. The subsequent mining is confined to each projected database, relevant to a smaller set of candidates. (2) The alternative-level projection in FreeSpan reduces the cost of scanning multiple projected databases when finding frequent length-3 candidates, which takes advantages of Apriori-like 3-way candidate filtering to produce smaller projected databases. Thus the method dramatically reduces the efforts of repeatedly generating and checking large sets of candidate sequences against the entire databases and achieves better performances when there exist a large set of sequential patterns.

Acknowledgements. Authors would like to express their thanks to Helen Pinto for her comments which help improve the quality of the paper.

References

[1] R. Agarwal, C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent itemsets. In *Journal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining)*, (to appear), 2000.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, September 1994.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering (ICDE'95)*, pages 3–14, Taipei, Taiwan, March 1995.

[4] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00)*, Dallas, TX, May 2000.

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

[6] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'96)*, pages 1–12, Montreal, Canada, June 1996.

[7] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. 5th Int. Conf. Extending Database Technology (EDBT)*, pages 3–17, Avignon, France, March 1996.