Approaches for Pattern Discovery Using Sequential Data Mining

Manish Gupta
University of Illinois at Urbana-Champaign, USA

Jiawei Han
University of Illinois at Urbana-Champaign, USA

ABSTRACT

In this chapter we first introduce sequence data. We then discuss different approaches for mining of patterns from sequence data, studied in literature. Apriori based methods and the pattern growth methods are the earliest and the most influential methods for sequential pattern mining. There is also a vertical format based method which works on a dual representation of the sequence database. Work has also been done for mining patterns with constraints, mining closed patterns, mining patterns from multi-dimensional databases, mining closed repetitive gapped subsequences, and other forms of sequential pattern mining. Some works also focus on mining incremental patterns and mining from stream data. We present at least one method of each of these types and discuss their advantages and disadvantages. We conclude with a summary of the work.

INTRODUCTION

What is Sequence Data?

Sequence data is omnipresent. Customer shopping sequences, medical treatment data, and data related to natural disasters, science and engineering processes data, stocks and markets data, telephone calling patterns, weblog click streams, program execution sequences, DNA sequences and gene expression and structures data are some examples of sequence data.

Notations and Terminology

Let I = {i_1, i_2, i_3 … i_n} be a set of **items**. An item-set X is a subset of items i.e. $X \subseteq I$. A **sequence** is an ordered list of **item-sets** (also called **elements** or **events**). Items within an element are unordered and we would list them alphabetically. 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**. E.g., s=<a(ce)(bd)(bcde)f(dg)> is a sequence which consists of 7 distinct items and 6 elements. Length of the sequence is 12.
A group of sequences stored with their identifiers is called a **sequence database**. We say that a sequence s is a **subsequence** of t, if s is a “projection” of t, derived by deleting elements and/or items from t. E.g. <a(c)(bd)f> is a subsequence of s. Further, sequence s is a **δ-distance subsequence** of t if there exist integers $j_1 < j_2 < … < j_n$ 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 ... n. That is, occurrences of adjacent elements of s within t are not separated by more than δ elements.

What is Sequential Pattern Mining?

Given a pattern p, **support** of the sequence pattern p is the number of sequences in the database containing the pattern p. A pattern with support greater than the support threshold min_sup is called a **frequent pattern** or a **frequent sequential pattern**. A sequential pattern of length l is called an **l-pattern**. **Sequential pattern mining** is the task of finding the complete set of frequent subsequences given a set of sequences. A huge number of possible sequential patterns are hidden in databases.
A sequential pattern mining algorithm should
a. find the complete set of patterns, when possible, satisfying the minimum support (frequency) threshold,
b. be highly efficient, scalable, involving only a small number of database scans
c. be able to incorporate various kinds of user-specific constraints.

APPROACHES FOR SEQUENTIAL PATTERN MINING

Apriori-Based Method (GSP: Generalized Sequential Patterns) (Srikant & Agrawal, 1996)

The Apriori property of sequences states that, if a sequence S is not frequent, then none of the super-sequences of S can be frequent. E.g, <hb> is infrequent implies that its super-sequences like <hab> and <(ah)b> would be infrequent too.

The GSP algorithm finds all the length-1 candidates (using one database scan) and orders them with respect to their support ignoring ones for which support < min_sup. Then for each level (i.e., sequences of length-k), the algorithm scans database to collect support count for each candidate sequence and generates candidate length-(k+1) sequences from length-k frequent sequences using Apriori. This is repeated until no frequent sequence or no candidate can be found.

Consider the database as shown in table 1. Our problem is to find all frequent sequences, given min_sup=2.

Table 1. Database.

Database	**Length-1 Patterns**

Seq Id	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

Cand	Seq
<a>	3
<b>	5
<c>	4
<d>	3
<e>	3
<f>	2
~~<g>~~	1
~~<h>~~	1

Length-2 Candidates

Table 2. Length-2 candidates

	<a>	<b>	<c>	<d>	<e>	<f>
<a>	<aa>	<ab>	<ac>	<ad>	<ae>	<af>
<b>	<ba>	<bb>	<bc>	<bd>	<be>	<bf>
<c>	<ca>	<cb>	<cc>	<cd>	<ce>	<cf>
<d>	<da>	<db>	<dc>	<dd>	<de>	<df>
<e>	<ea>	<eb>	<ec>	<ed>	<ee>	<ef>
<f>	<fa>	<fb>	<fc>	<fd>	<fe>	<ff>

	<a>	<b>	<c>	<d>	<e>	<f>
<a>		<(ab)>	<(ac)>	<(ad)>	<(ae)>	<(af)>
<b>			<(bc)>	<(bd)>	<(be)>	<(bf)>
<c>				<(cd)>	<(ce)>	<(cf)>
<d>					<(de)>	<(df)>
<e>						<(ef)>
<f>						

As shown in Table 2, using Apriori one needs to generate just 51 length-2 candidates, while without Apriori property, 8*8+8*7/2=92 candidates would need to be generated. For this example, Apriori would perform 5 database scans, pruning away candidates with support less than min_sup. Candidates that cannot pass support threshold are pruned.
1st scan: 8 candidates. 6 length-1 sequence patterns.
2nd scan: 51 candidates. 19 length-2 sequence patterns. 10 candidates not in DB at all
3rd scan: 46 candidates. 19 length-3 sequence patterns. 20 candidates not in DB at all
4th scan: 8 candidates. 6 length-4 sequence patterns.
5th scan: 1 candidate. 1 length-5 sequence patterns.

Some drawbacks of GSP are: a huge set of candidate sequences are generated, multiple scans of database are needed and it is inefficient for mining long sequential patterns (as it needs to generate a large number of small candidates).

Apart from finding simple frequent patterns, GSP generalizes the problem by

a. Allowing a user to specify time constraints (minimum and/or maximum time period between adjacent elements in a pattern)
b. Relaxing the restriction that the items in an element of a sequential pattern must come from the same transaction, instead allowing the items to be present in a set of transactions whose transaction-times are within a user-specified time window.

c. Given a user-defined taxonomy (is-a hierarchy) on items, allowing sequential patterns to include items across all levels of the taxonomy.

Vertical Format-Based Method (SPADE: Sequential PAttern Discovery using Equivalent Class) (Zaki, 2001)

This is a vertical format sequential pattern mining method. SPADE first maps the sequence database to a vertical id-list database format which is a large set of items <SID (Sequence ID), EID (Event ID)>. Sequential pattern mining is performed by growing the subsequences (patterns) one item at a time by Apriori candidate generation.

As shown in table 3 below, all frequent sequences can be enumerated via simple temporal joins (or intersections) on id-lists. They use a lattice-theoretic approach to decompose the original search space (lattice) into smaller pieces (sub-lattices) which can be processed independently in main-memory.

Their approach usually requires three database scans, or only a single scan with some pre-processed information, thus minimizing the I/O costs. SPADE decouples the problem decomposition from the pattern search. Pattern search could be done in a BFS (breadth first search) or a DFS (depth first search) manner. The vertical id-list based approach is also insensitive to data-skew. It also has linear scalability with respect to the number of input-sequences, and a number of other database parameters.

Table 3. Frequent sequences

SID	EID	Items
1	1	a
1	2	abc
1	3	ac
1	4	d
1	5	cf
2	1	ad
2	2	c
2	3	bc
2	4	ae
3	1	ef
3	2	ab
3	3	df
3	4	c
3	5	b

a		b		…
SID	EID	SID	EID	…
1	1	1	2	
1	2	2	3	
1	3	3	2	
2	1			
3	2			

ab			ba			…
SID	EID(a)	EID(b)	SID	EID(b)	EID(a)	…
1	1	2	1	2	3	
2	1	3				

aba			
SID	EID(a)	EID(b)	EID(a)
1	1	2	3

Pattern Growth Based Methods

FreeSpan (Han, Pei, Asl, Chen, Dayal, & Hsu, 2000) & PrefixSpan (Pei, et al., 2001)

These methods help in avoiding the drawbacks of the Apriori based methods.

FreeSpan (Frequent pattern projected Sequential pattern mining) uses frequent items to recursively project sequence databases into a set of smaller projected databases and grows 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.

FreeSpan first scans the database, collects the support for each item, and finds the set of frequent items. Frequent items are listed in support descending order (in the form of item:support) E.g., flist=a:4, b:4, c:4, d:3, e:3, f:3.

According to flist, the complete set of sequential patterns in S can be divided into 6 disjoint subsets: (1) the ones containing only item ‘a’, (2) the ones containing item ‘b’, but containing no items after ‘b’ in flist, (3) the ones containing item ‘c’, but no items after ‘c’, in flist, and so on, and finally, (6) ones containing item ‘f’.

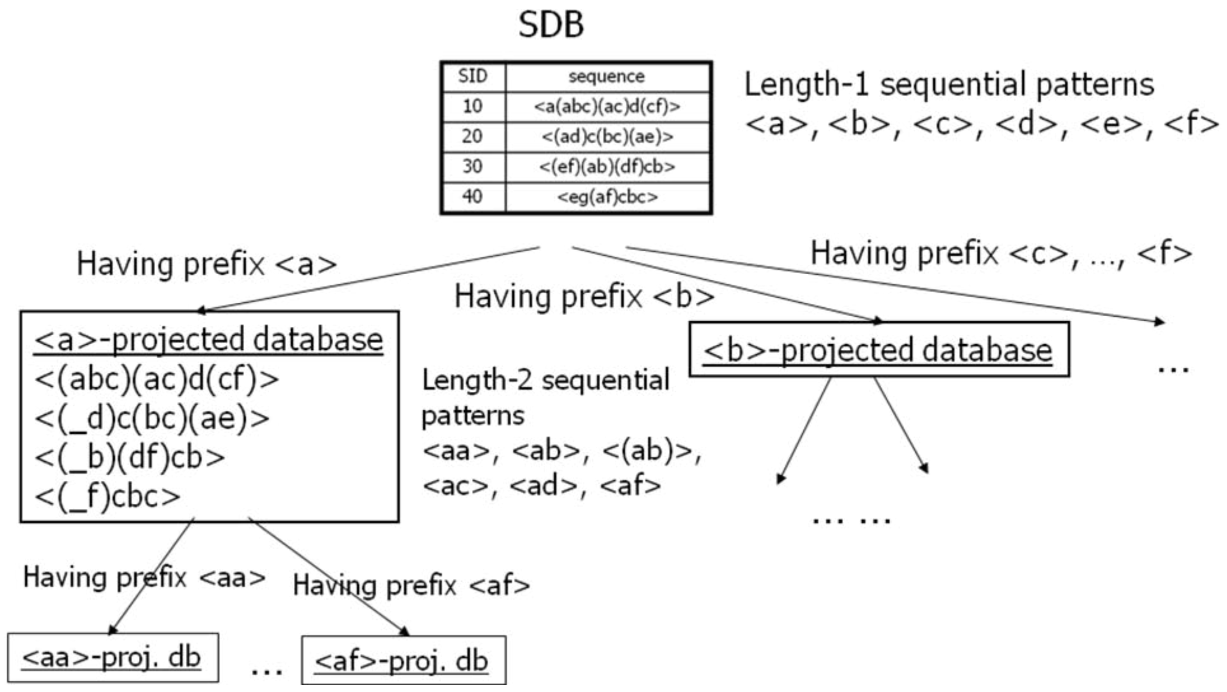
The subsets of sequential patterns can be mined by constructing projected databases. Infrequent items, such as ‘g’ in this example, are removed from construction of projected databases.

Note that {b}, {c}, {d}, {e}, {f}-projected databases are constructed simultaneously during one scan of the original sequence database. All sequential patterns containing only item ‘a’ are also found in this pass. This process is performed recursively on projected databases. Since FreeSpan projects a large sequence database recursively into a set of small projected sequence databases based on the currently mined frequent sets, the subsequent mining is confined to each projected database relevant to a smaller set of candidates.

The major cost of FreeSpan is to deal with projected databases. If a pattern appears in each sequence of a database, its projected database does not shrink (except for the removal of some infrequent items). Moreover, since a length-k subsequence may grow at any position, the search for length-(k+1) candidate sequence will need to check every possible combination, which is costly.

PrefixSpan (Prefix-projected Sequential pattern mining) works similar to FreeSpan except that the partitioning is done using **prefixes** of sequences. E.g., for a sequence <(abc)(ac)d(cf)>, <ab> is a prefix which has <(_c)(ac)d(cf)> as the corresponding suffix (projection) as shown in Figure 1.

Figure 1. PrefixSpan



Its general idea is to examine only the frequent prefix subsequences and project only their corresponding postfix subsequences into projected databases because any frequent subsequence can always be found by growing a frequent prefix. Thus the search space for our example will be partitioned into the following six subsets according to the six prefixes: (1) the ones having prefix <a> ... and (6) the ones having prefix <f>. In each projected database, sequential patterns are grown by exploring only local frequent patterns. The subsets of sequential patterns can be mined by constructing corresponding projected databases and mining each recursively.

PrefixSpan first finds sequential patterns having prefix <a>. Recursively, all sequential having patterns prefix <a> can be partitioned into 6 subsets: (1) those having prefix <aa> (2) those having prefix <ab>… and finally, (6) those having prefix <af>. These subsets can be mined by constructing respective projected databases (only if the prefix is frequent) and mining each recursively. Similarly, we can find sequential patterns having prefix <b>, <c>, <d>, <e> and <f> respectively, by constructing <b>-, <c>-, <d>-, <e>- and <f>-projected databases and mining them respectively.

No candidate sequence needs to be generated by PrefixSpan. Projected databases keep shrinking. The major cost of PrefixSpan is the construction of projected databases. To further improve mining efficiency, two kinds of database projections are explored: level-by-level projection and bi-level projection. Moreover, a main-memory-based pseudo-projection (using pointers rather than physically copying postfix sequences) technique is developed for saving the cost of projection and speeding up processing when the projected (sub)-database and its associated pseudo-projection processing structure can fit in main memory. PrefixSpan mines complete set of patterns much faster than both GSP and FreeSpan.

Constraint Based Methods

Conventionally, users can specify only min_sup as a parameter to a sequential pattern mining algorithm. There are two major difficulties in sequential pattern mining: (1) effectiveness: the mining may return a huge number of patterns, many of which could be uninteresting to users, and (2) efficiency: it often takes substantial computational time and space for mining the complete set of sequential patterns in a large sequence database. To prevent these problems, users can use constraint based sequential pattern mining for focused mining of desired patterns. Constraints could be anti-monotone, monotone, succinct, convertible or inconvertible. Anti-monotonicity means "if an item-set does not satisfy the rule constraint, then none of its supersets satisfy". Monotonicity means "if an item-set satisfies the rule constraint, then all of its supersets satisfy". Succinctness means "All and only those patterns guaranteed to satisfy the rule can be enumerated". Convertible constraints are those which are not any of anti-monotonic, monotonic, succinct but can be made anti-monotonic or monotonic constraints by changing order of elements in the set. Inconvertible constraints are the ones which are not convertible.

In the context of constraint-based sequential pattern mining, **(Srikant & Agrawal, 1996)** generalized the scope of the Apriori-based sequential pattern mining to include time constraints, sliding time windows, and user-defined taxonomy. Mining frequent episodes in a sequence of events studied by **(Mannila, Toivonen, & Verkamo, 1997)** can also be viewed as a constrained mining problem, since episodes are essentially constraints on events in the form of acyclic graphs. The classical framework on frequent and sequential pattern mining is based on the anti-monotonic Apriori property of frequent patterns. A breadth-first, level-by-level search can be conducted to find the complete set of patterns.

Performance of conventional constraint-based sequential pattern mining algorithms dramatically degrades in the case of mining long sequential patterns in dense databases or when using low minimum supports. In addition, the algorithms may reduce the number of patterns but unimportant patterns are still found in the result patterns. **(Yun, 2008)** uses weight constraints to reduce the number of unimportant patterns. During the mining process, they consider not only supports but also weights of patterns. Based on the framework, they present a weighted sequential pattern mining algorithm (WSpan).

(Chen, Cao, Li, & Qian, 2008) incorporate user-defined tough aggregate constraints so that the discovered knowledge better meets user needs. They propose a novel algorithm called PTAC (sequential frequent Patterns mining with Tough Aggregate Constraints) to reduce the cost of using tough aggregate constraints by incorporating two effective strategies. One avoids checking data items one by one by utilizing the features of "promising-ness" exhibited by some other items and validity of the corresponding prefix. The other avoids constructing an unnecessary projected database by effectively pruning those unpromising new patterns that may, otherwise, serve as new prefixes.

(Masseglia, Poncelet, & Teisseire, 2003) propose an approach called GTC (Graph for Time Constraints) for mining time constraint based patterns (as defined in GSP algorithm) in very large databases. It is based on the idea that handling time constraints in the earlier stage of the data mining process can be highly beneficial. One of the most significant new features of their approach is that handling of time constraints can be easily taken into account in traditional level-wise approaches since it is carried out prior to and separately from the counting step of a data sequence.

(Wang, Chirn, Marr, Shapiro, Shasha, & Zhang, 1994) looked at the problem of discovering approximate structural patterns from a genetic sequences database. Besides the minimum support threshold, their solution allows the users to specify: 1. the desired form of patterns as sequences of consecutive symbols separated by variable length don't cares, 2. a lower bound on the length of the discovered patterns, and 3. an upper bound on the edit distance allowed between a mined pattern and the data sequence that contains it. Their algorithm uses a random sample of the input sequences to build a main memory data structure, termed generalized suffix tree, that is used to obtain an initial set of candidate pattern segments and screen out candidates that are unlikely to be frequent based on their occurrence counts in the sample. The entire database is then scanned and filtered to verify that the remaining candidates are indeed frequent answers to the user query.

(Garofalakis, Rastogi, & Shim, 2002) propose regular expressions as constraints for sequential pattern mining and developed a family of **SPIRIT (Sequential pattern mining with regular expression constraints)** algorithms. Members in the family achieve various degrees of constraint enforcement. The algorithms use relaxed constraints with nice properties (like anti-monotonicity) to filter out some unpromising patterns/candidates in their early stage. A SPIRIT algorithm first identifies C' as a constraint weaker than C. Then it obtains F_1=frequent items in D that satisfy C'. Further, it iteratively generates candidates C_k using F and C', prunes candidates in C_k that contain subsequences that satisfy C' but are not in F, identifies F_k as the frequent sequences in C_k by scanning the database to count support and updates F to $F \cup F_k$. Finally, sequences in F that satisfy the original condition C are output.

General SPIRIT constrained mining framework can be specified as:

```
PROCEDURE SPIRIT(D,C)
Begin
    1. Let C'=a constraint weaker (i.e., less restrictive) than C.
    2. F=$F_1$=frequent items in D that satisfy C'
    3. K=2
    4. Repeat {
        a. //candidate generation
        b. Using C' and F generate $C_k$={potentially frequent k-sequences that satify
           C'}
        c. //candidate pruning
        d. Let P={s∈$C_k$: s has a subsequence t that satisfies C' and t∉F}
        e. $C_k$=$C_k$-P
        f. //candidate counting
        g. Scan D counting the support for candidate k-sequences in $C_k$
        h. $F_k$-frequent sequences in $C_k$
        i. F=F∪$F_k$
        j. K=K+1
5. }until TerminatingCondition(F < C') holds
6. //enforce the original (stronger) constraint C
7. Output sequences in F that satisfy C
8. End
```

Given a user specified RE constraint C, the first SPIRIT algorithm SPIRIT(N) ("N" for "Naive") only prunes candidate sequences containing elements that do not appear in C. The second one, SPIRIT(L) ("L" for "Legal"), requires every candidate sequence to be legal with respect to some state of automata A(C). The third, SPIRIT(V) ("V" for "Valid"), filters out candidate sequences that are not valid with respect to any state of A(C). The fourth, SPIRIT(R) ("R" for "Regular"), pushes C all the way inside the mining process by counting support only for valid candidate sequences.

The above interesting studies handle a few scattered classes of constraints. However, two problems remain. First, many practical constraints have not been covered. Also there is a need for a systematic method to push various constraints into the mining process. Unfortunately, some commonly encountered sequence-based constraints, such as regular expression constraints, are neither monotonic, nor anti-monotonic, nor succinct. **(Pei, Han, & Wang, 2007)** mention seven categories of constraints:
1. Item constraint: An item constraint specifies subset of items that should or should not be present in the patterns.
2. Length constraint: A length constraint specifies the requirement on the length of the patterns, where the length can be either the number of occurrences of items or the number of transactions.
3. Super-pattern constraint: Super-patterns are ones that contain at least one of a particular set of patterns as sub-patterns.
4. Aggregate constraint: An aggregate constraint is the constraint on an aggregate of items in a pattern, where the aggregate function can be sum, avg, max, min, standard deviation, etc.
5. Regular expression constraint: A regular expression constraint CRE is a constraint specified as a regular expression over the set of items using the established set of regular expression operators, such as disjunction and Kleene closure.
6. Duration constraint: A duration constraint is defined only in sequence databases where each transaction in every sequence has a time-stamp. It requires that the sequential patterns in the sequence database must have the property such that the time-stamp difference between the first and the last transactions in a sequential pattern must be longer or shorter than a given period.

7. Gap constraint: A gap constraint set is defined only in sequence databases where each transaction in every sequence has a timestamp. It requires that the sequential patterns in the sequence database must have the property such that the timestamp difference between every two adjacent transactions must be longer or shorter than a given gap.

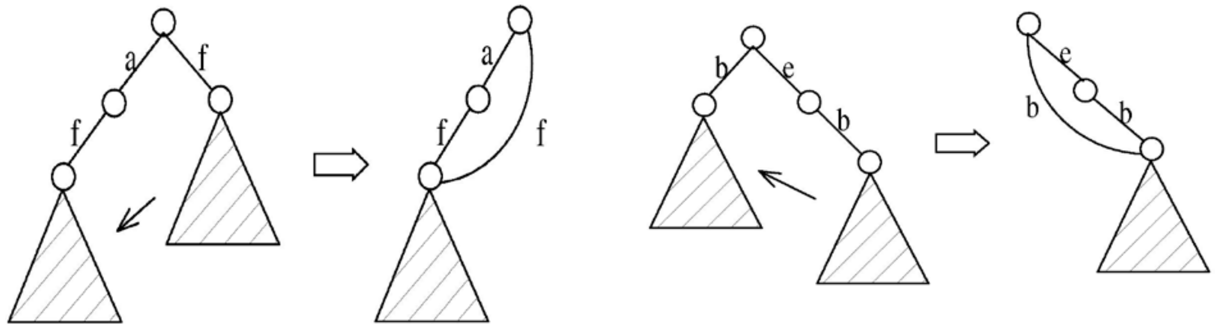
A constraint C_{pa} is called prefix anti-monotonic if for each sequence 'a' satisfying the constraint, so does every prefix of 'a'. A constraint C_{pm} is called prefix monotonic if for each sequence 'a' satisfying the constraint, so does every sequence having 'a' as a prefix. A constraint is called prefix-monotone if it is prefix anti-monotonic or prefix monotonic.

The authors describe a **pattern-growth (PG) method for Constraint-based sequential pattern mining** which is based on a prefix-monotone property. They show that all the monotonic and anti-monotonic constraints, as well as regular expression constraints, are prefix-monotone, and can be pushed deep into a PG-based mining. Moreover, some tough aggregate constraints, such as those involving average or general sum, can also be pushed deep into a slightly revised PG mining process. In the recursive FP growth framework, the authors first compute all the length-1 frequent prefixes. Then they compute the corresponding projected databases. Each of the frequent prefixes of length (l+1) are further processed recursively only if they satisfy the constraint C.

Closed Sequential Pattern Mining

CloSpan (Yan, Han, & Afshar, 2003) is an algorithm for the mining of closed repetitive gapped subsequences (figure 2). A closed sequential pattern s is a sequence such that there exists no super-pattern s', s '⊃ s, and s' and s have the same support. E.g., given <abc>: 20, <abcd>:20, <abcde>: 15, we know that <abcd> is closed. If the database contains 1 long sequence with 100 elements and min support is 1, this sequence will generate 2^100 frequent subsequences, though there is only one of these which is closed. Mining of closed sequences reduces the number of (redundant) patterns but attains the same expressive power. Note that if s' ⊃ s, s is closed iff two projected DBs have the same size. CloSpan uses backward sub-pattern and backward super-pattern pruning to prune redundant search space thereby preventing unnecessary computations.

Figure 2. CloSpan



Backward super-pattern pruning **Backward sub-pattern pruning**

CloSpan is basically similar to PrefixSpan with sub-pattern and super-pattern checks which involve checking and matching of the size of the databases. The authors show that CloSpan performs better than PrefixSpan in terms of execution time.

Sequential Pattern Mining in Data Streams: SS-BE and SS-MB (Mendes, Ding, & Han, 2008)

Data stream is an unbounded sequence in which new elements are generated continuously. Memory usage is limited and an algorithm is allowed to perform only a single scan over the database. Two effective methods for stream-based sequential pattern mining are SS-BE (Stream Sequence miner using Bounded Error) and SS-MB (Stream Sequence miner using Memory Bounds).

SS-BE Method can be outlined as follows:

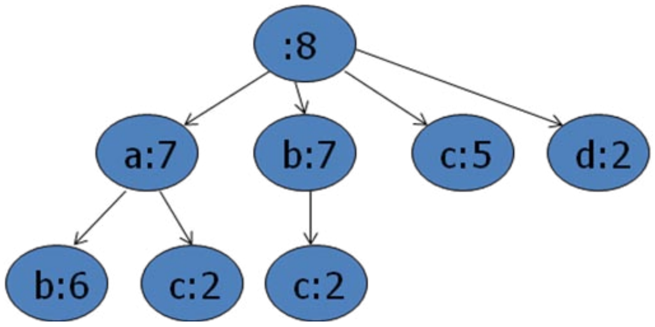
a. Break the stream into fixed-sized batches.
b. For each arriving batch, apply PrefixSpan. Insert each frequent sequence found into a tree.
c. Periodically prune the tree (the number of batches seen is a multiple of the pruning period).
d. Output all sequences corresponding to nodes having count >= $(\sigma-\epsilon)N$.

This method outputs no false negatives and true support of false positives is at least $(\sigma-\epsilon)$.
E.g., suppose $\sigma = 0.75$, $\epsilon = 0.5$ and data stream D: <a,b,c>, <a,c>, <a,b>, <b,c>, <a,b,c,d>, <c,a,b>, <d,a,b>, <a,e,b>. Let the first batch B_1 contain the first four sequences and the second batch B_2 contain the next four. The algorithm first applies PrefixSpan to B_1 with min_sup as 0.5. The frequent sequences found are: <a>:3, <b>:3, <c>:3, <a,b>:2, <a,c>:2, and <b,c>:2. A frequent pattern tree is created. Let the pruning period be two batches. So algorithm proceeds to batch B_2. The frequent sequences found are: <a>:4, <b>:4, <c>:2, <d>:2, and <a,b>:4. The frequent pattern tree would look as shown in the figure below. Now SS-BE would prune the tree by identifying and removing all nodes guaranteed to have true support below $\epsilon = 0.5$ during the time they were kept in the tree. Thus <d>:2, <ac>:2 and <bc>:2 are pruned away.

Finally SS-BE outputs all sequences having count at least $(\sigma-\epsilon)N = (0.75 - 0.5)*8 = 2$. Thus output is <a>: 7, <b>: 7, <c>: 5, <a, b>:6. Note that there are no false negatives and only one false positive: <c>.

SS-MB method is similar to SS-BE except that in step 3, rather than pruning the tree after a time period, the tree size is limited to 'm' nodes. Due to this, SS-MB can only guarantee no false negatives after execution. E.g. in the above example, assume that 'm' is 7. Then after batch B_2 is processed, the tree contains 8 nodes and hence the node with minimum support <b,c> is removed (figure 3). Because of the specific 'm', SS-MB can control amount of memory used explicitly.

Figure 3. SS-BE pruning tree



The authors show that the two methods are effective solutions to the stream sequential pattern mining problem: running time scales linearly, maximum memory usage is limited and a very small number of false positives are generated.

Mining Incremental Patterns: IncSpan (Incremental Mining of Sequential Patterns) (Cheng, Yan, & Han, 2004)

Many real life sequence databases, such as customer shopping sequences, medical treatment sequences, etc., grow incrementally. It is undesirable to mine sequential patterns from scratch each time when a small set of sequences grow, or when some new sequences are added into the database. Incremental algorithm should be developed for sequential pattern mining so that mining can be adapted to frequent and incremental database updates, including both insertions and deletions. However, it is nontrivial to mine sequential patterns incrementally, especially when the existing sequences grow incrementally because such growth may lead to the generation of many new patterns due to the interactions of the growing subsequences with the original ones. There are two kinds of database updates in applications: (1) inserting new sequences (INSERT) and (2) appending new item-sets/items to the existing sequences (APPEND). Let DB be the old database, Δdb be the change and DB' be the new database. Thus, DB' = DB ∪Δdb.

It is easier to handle the first case: INSERT. An important property of INSERT is that a frequent sequence in DB' = DB ∪Δdb must be frequent in either DB or Δdb (or both). If a sequence is infrequent in both DB and Δdb, it cannot be frequent in DB'. Thus, only those patterns that are frequent in Δdb but infrequent in DB need to be searched in DB to find their occurrence count. (Zhang, Kao, Cheung, & Yip, 2002) propose another algorithm of incremental mining to handle the case of INSERT in sequential pattern mining.

For the second case, consider that new items only get appended. Suppose |DB|=1000 and |Δdb|=20, min_sup=10%. Suppose a sequence ‘s’ is infrequent in DB with 99 occurrences (sup = 9:9%). In addition, it is also infrequent in Δdb with only 1 occurrence (sup = 5%). Although ‘s’ is infrequent in both DB and Δdb, it becomes frequent in DB' with 100 occurrences.

This problem complicates the incremental mining since one cannot ignore the infrequent sequences in Δdb, but there are an exponential number of infrequent sequences even in a small Δdb and checking them against the set of infrequent sequences in DB will be very costly. (Parthasarathy, Zaki, Ogihara, & Dwarkadas, 1999) proposed an incremental mining algorithm, called ISM, based on SPADE by exploiting a concept called negative border. However, maintaining negative border is memory consuming and not well adapted for large databases. (Masseglia, Poncelet, & Teisseire, Efficient mining of sequential patterns with time constraints: Reducing the combinations, 2009) developed another incremental mining algorithm using candidate generate-and-test approach, which is costly, especially when the sequences are long because it requires multiple scans of the whole database.

For the third case, where the database is updated with both INSERT and APPEND, the problem becomes even more complicated. There are two approaches: (1) handling them separately by first performing APPEND then INSERT; (2) treat the inserted sequences as appending to empty sequences in DB: a special case of APPEND. Then this problem is reduced to APPEND.

Given a minimum support threshold, min_sup, a sequence is frequent if its support >=min_sup; given a factor μ<=1, a sequence is semi-frequent if its support<min_sup but >μ*min_sup; a sequence is infrequent if its support<μ*min_sup. Let FS be the set of all frequent sequential patterns and SFS be the set of semi-frequent sequential patterns.

Given a sequence database DB, min_sup, the set of frequent subsequences FS in DB, and an appended sequence database DB’ of D, the problem of incremental sequential pattern mining is to mine the set of frequent subsequences FS’ in DB’ based on FS instead of mining on DB’ from scratch. A simple algorithm, SimpleSpan, exploits the FS in the original database and incrementally mines new patterns. SimpleSpan updates the support of every frequent sequence in FS, adds it to FS’ and uses it as a prefix to project database. In addition, SimpleSpan scans the new database DB’ to discover new frequent single items and uses them as prefix to project database using PrefixSpan. One problem of SimpleSpan is that it makes a large number of database projections, which is costly. The drawback of SimpleSpan is that it has no information about infrequent sequences in the original database DB. But such information can enable us to reduce search space and find new frequent sequences efficiently.

IncSpan uses the technique of buffering semi-frequent patterns by maintaining a set SFS in the original database DB. Since the sequences in SFS are “almost frequent”, most of the frequent subsequences in the appended database will either come from

SFS or they are already frequent in the original database. With a minor update to the original database, it is expected that only a small fraction of subsequences which were infrequent previously would become frequent. This is based on the assumption that updates to the original database have a uniform probability distribution on items. It is expected that most of the frequent subsequences introduced by the updated part of the database would come from the SFS. The SFS forms a kind of boundary (or "buffer zone") between the frequent subsequences and infrequent subsequences.

IncSpan algorithm can be outlined as follows.

a. Scan Δdb for single items. If a new item or an infrequent item becomes frequent or semi-frequent, add it to FS' or SFS'. For every item in FS', use it as prefix to construct projected database and discover frequent sequences recursively.
b. Check every pattern in FS and SFS in Δdb to adjust the support of those patterns.
    I. If a pattern becomes frequent, add it to FS'. Then check whether it meets the projection condition. If so, use it as prefix to project database. Discover frequent or semi-frequent patterns in the projected database. To improve the performance, shared projection can be used in this step.
    II. If a pattern is semi-frequent, add it to SFS'.

The authors also mention two optimization techniques, reverse pattern matching and shared projection to improve the performance.

Multidimensional Sequential Pattern Mining: UNISEQ (Pinto, Han, Pei, Wang, Chen, & Dayal, 2001)

Consider pattern P1= {try a 100 hour free internet access package⇒subscribe to 15 hours/month package⇒upgrade to 30 hours per month package⇒upgrade to unlimited package}. This pattern may hold for all customers below age of 35 (75% customers). But for other customers, pattern P2= {try a 100 hour free internet access package⇒ upgrade to 30 hours per month package} may hold. Clearly, if sequential pattern mining can be associated with customer category or other multi-dimensional information, it will be more effective since the classified patterns are often more useful. (Pinto, Han, Pei, Wang, Chen, & Dayal, 2001) propose two categories of methods: a. integration of efficient sequential pattern mining and multi-dimensional analysis methods (Seq-Dim and Dim-Seq). b. embedding multi-dimensional information into sequences and mine the whole set using a uniform sequential pattern mining method (Uni-Seq).

A multi-dimensional sequence database has the schema (RID, A_1, A_2 … A_m, S) where RID is the record identifier, A_1 … A_m are the attributes and S is the sequence. A multi-dimensional pattern 'p' would match a tuple 't' in the database, if the attribute values match (or the attribute value is *) and 's' is a subsequence of the sequence stored in 't'. e.g. t=(10, business, Boston, middle, <(bd)cba>)

UniSeq (Uniform Sequential): Multi-dimensional information in a tuple 't' in multi-dimensional DB can be embedded in the sequence by introducing a special element. E.g. 't' can be rewritten as (10, <(business Boston middle)(bd)cba>). Let the database containing such modified tuples be called MD-extension DB and denoted as SDB-MD. Now the problem is: Given, SDB-MD and min_sup, output the complete set of multi-dimensional sequential patterns. UniSeq mines sequential patterns in SDB-MD using PrefixSpan. For each sequential pattern 'p' in SDB-MD, it outputs the corresponding multi-dimensional sequential pattern in SDB. As an alternative, instead of embedding the multi-dimensional information into the first element of each sequence, it can be attached as the last element. Both the alternatives have almost identical performance results. Thus, UniSeq reduces the problem to mining one extended sequence database and is therefore easy to implement. But, all dimension values are treated as sequential items. Hence, it cannot take advantage of efficient mining algorithms for multi-dimensional non-sequential computational methods. Hence, cost of computing becomes high when data has large number of dimensions.

A SDB-MD can be partitioned into two parts: dimensional information and sequence. So, we can first mine patterns about dimensional information (called multi-dimensional patterns or MD-patterns) and then find sequential patterns from projected sub-database (tuples containing the MD-pattern) or vice versa. **Dim-Seq** first finds MD-patterns and then for each MD-pattern, it forms MD-projected database and mines sequential patterns in projected databases. **Seq-Dim** first mines the sequential patterns. For each sequential pattern, it forms projected MD-database and then finds MD-patterns within projected databases. Seq-Dim is more efficient and scalable in general compared to Dim-Seq.

Mining Closed Repetitive Gapped Subsequences (Ding, Lo, Han, & Khoo, 2009)

Patterns often repeat multiple times in a sequence e.g., in program execution traces, sequences of words (text data), credit card usage histories. Given two sequences like S_1 = AABCDABB, S_2 = ABCD, is pattern AB more frequent then CD? To answer this question, one needs to define a notion of repetitive support, sup(P) as max{|INS|: INS is a set of non-overlapping instances of P}. The aim is to maximize the size of the non-overlapping instance set. Note that if P' is a super-pattern of P, then sup(P') ≤ sup(P).

To solve this problem, the authors propose a greedy instance-growth algorithm. The intuition is to extend each instance to the nearest possible event. Consider a database of two sequences as shown in table 4:

Table 4. Database of two sequences

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
S1	A	B	C	A	C	B	D	D	B
S2	A	C	D	B	A	C	A	D	D

Support set I^A	**Support set I^{AC}**	**Support set I^{ACB}**
(1,<1>)	**(1,<1,3>)**	**(1,<1,3,6>)**
(1,<4>)	**(1,<4,5>)**	**(1,<4,5,9>)**
(2,<1>)	**(2,<1,2>)**	**(2,<1,2,4>)**
(2,<5>)	**(2,<5,6>)**	
(2,<7>)		
sup(A)=5	**sup(AC)=4**	**sup(ACB)=3**

The algorithm uses a procedure INSgrow(P, INS, e) which does the following. Given a leftmost support set INS of P, with |INS| = sup(P), and event e, it extends each instance in INS to the nearest possible event e and returns a support set INS^+ of pattern P○e (P concatenated with e). Thus, using this method, one can find all the frequent patterns by doing DFS in the pattern space.

Further, they define pattern extension as set of patterns with one more event. E.g., if P $=e_1e_2 \ldots e_m$, PExtension(P, e) = $\{ee_1e_2\ldots e_m, e_1ee_2\ldots e_m, \ldots, e_1e_2\ldots e_me\}$. Pattern P is not closed iff sup(P) = sup(Q) for some Q ∈ Extension(P, e). Also note that it is possible that AB is not closed but ABAC is closed. To prune the search space, they propose the following instance-border checking principle. Pattern P is prunable if there exists Q ∈ Extension(P, e) for some e such that sup(P) = sup(Q) (P is not closed) and for each (i, $<k_1, k_2, \ldots, k_{|P|}>$) ∈ INS^P and (i, $<k_1', k_2', \ldots, k_{|Q|}'>$) ∈ $INS^{Q:}$ $k_{|Q|}' \leq k_{|P|}$ where INS^P and INS^Q are (leftmost) support sets of P and Q respectively.

OTHER SEQUENTIAL PATTERN MINING METHODS

(Kum, Chang, & Wang, Sequential Pattern Mining in Multi-Databases via Multiple Alignment, 2006) proposed a new sequential pattern mining method based on multiple alignment (rather than the usual support-based approach) for mining multiple databases. Multiple databases are mined and summarized at the local level, and only the summarized patterns are used in the global mining process. For summarization, they propose the theme of approximate sequential pattern mining roughly defined as identifying patterns approximately shared by many sequences. They propose an algorithm, ApproxMAP, to mine approximate sequential patterns, called consensus patterns, from large sequence databases in two steps. First, sequences are clustered by similarity. Then, consensus patterns are mined directly from each cluster through multiple alignment.

Further, **(Kum, Chang, & Wang, Benchmarking the effectiveness of sequential pattern mining methods, 2007)** benchmarked the effectiveness of sequential pattern mining methods by comparing a support-based sequential pattern model with an approximate pattern model based on sequence alignment using a metric that evaluates how well a mining method finds known common patterns in synthetic data. Their comparison study suggests that the alignment model will give a good summary of the sequential data in the form of a set of common patterns in the data. In contrast, the support model generates massive amounts of frequent patterns with much redundancy. This suggests that the results of the support model require more post processing before it can be of actual use in real applications.

(Laur, Symphor, Nock, & Poncelet, 2007) introduced statistical supports to maximize mining precision and improve the computational efficiency of the incremental mining process. As only a part of the stream can be stored, mining data streams for sequential patterns and updating previously found frequent patterns need to cope with uncertainty. They introduce a new statistical approach which biases the initial support for sequential patterns. This approach holds the advantage to maximize either the precision or the recall, as chosen by the user, and limit the degradation of the other criterion. Moreover, these statistical supports help building statistical borders which are the relevant sets of frequent patterns to use into an incremental mining process.

(Lin, Chen, Hao, Chueh, & Chang, 2008) introduced the notion of positive and negative sequential patterns, where positive patterns include the presence of an item-set of a pattern, and negative patterns are the ones with the absence of an item-set.

Items sold in a store can usually be organized into a concept hierarchy according to some taxonomy. Based on the hierarchy, sequential patterns can be found not only at the leaf nodes (individual items) of the hierarchy, but also at higher levels of the hierarchy; this is called multiple-level sequential pattern mining. In previous research, taxonomies had crisp relationships between the categories in one level and the categories in another level. In real life, however, crisp taxonomies cannot handle the uncertainties and fuzziness inherent in the relationships among items and categories. For example, the book Alice's Adventures in Wonderland can be classified into the Children's Literature category, but can also be related to the Action & Adventure category. To deal with the fuzzy nature of taxonomy, **(Chen & Huang, A novel knowledge discovering model for mining fuzzy multi-level sequential patterns in sequence databases, 2008)** apply fuzzy set techniques to concept taxonomies so that the relationships from one level to another can be represented by a value between 0 and 1. They propose a fuzzy multiple- level mining algorithm (FMSM) to extract fuzzy multiple-level sequential patterns from databases. In addition, another algorithm, named the CROSS-FMSM algorithm, is developed to discover fuzzy cross-level sequential patterns.

(Kuo, Chao, & Liu, 2009) use K-means algorithm to achieve better computational efficiency for fuzzy sequential pattern mining.

Many methods only focus on the concept of frequency because of the assumption that sequences' behaviors do not change over time. The environment from which the data is generated is often dynamic; the sequences' behaviors may change over time. To adapt the discovered patterns to these changes, **(Chen & Hu, Constraint-based sequential pattern mining: the consideration of recency and compactness, 2006)** introduce two new concepts, recency and compactness and incorporate them into traditional sequential pattern mining. The concept of recency causes patterns to quickly adapt to the latest behaviors in sequence databases, while the concept of compactness ensures reasonable time spans for the discovered patterns. An efficient method is presented to find CFR-patterns (compactness, frequency, and recency).

CONCLUSION

We discussed basics of sequential pattern mining. We presented an exhaustive survey of different sequential pattern mining methods proposed in the literature. Sequential pattern mining methods have been used to analyze this data and identify patterns. Such patterns have been used to implement efficient systems that can recommend based on previously observed patterns, help in making predictions, improve usability of systems, detect events and in general help in making strategic product decisions. We envision that the power of sequential mining methods has not yet been fully exploited. We hope to see many more strong applications of these methods in a variety of domains in the years to come. Apart from this, new sequential pattern mining methods may also be developed to handle special scenarios of colossal patterns, approximate sequential patterns and other kinds of sequential patterns specific to the applications.

REFERENCES

Chen, E., Cao, H., Li, Q., & Qian, T. (2008). Efficient strategies for tough aggregate constraint-based sequential pattern mining. *Inf. Sci., 178*(6), 1498-1518.

Chen, Y.-L., & Hu, Y.-H. (2006). Constraint-based sequential pattern mining: The consideration of recency and compactness. *Decis. Support Syst., 42*(2), 1203-1215.

Chen, Y.-L., & Huang, T. C.-K. (2008). A novel knowledge discovering model for mining fuzzy multi-level sequential patterns in sequence databases. *Data Knowl. Eng., 66*(3), 349-367.

Cheng, H., Yan, X., & Han, J. (2004). IncSpan: Incremental mining of sequential patterns in large database. *KDD '04: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (pp. 527-532).

Ding, B., Lo, D., Han, J., & Khoo, S.-C. (2009). *Efficient mining of closed repetitive gapped subsequences from a sequence database*. ICDE 09.

Exarchos, T. P., Tsipouras, M. G., Papaloukas, C., & Fotiadis, D. I. (2008). A two-stage methodology for sequence classification based on sequential pattern mining and optimization. *Data Knowl. Eng., 66*(3), 467-487.

Garofalakis, M., Rastogi, R., & Shim, K. (2002). Mining sequential patterns with regular expression constraints. *IEEE Trans. on Knowl. and Data Eng., 14*(3), 530-552.

Han, J., Pei, J., Asl, B. M., Chen, Q., Dayal, U., & Hsu, M. C. (2000). FreeSpan: Frequent pattern-projected sequential pattern mining. *KDD '00: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 355-359). Boston, MA: ACM.

Kum, H.-C., Chang, J. H., & Wang, W. (2007). Benchmarking the effectiveness of sequential pattern mining methods. *Data Knowl. Eng., 60*(1), 30-50.

Kum, H.-C., Chang, J. H., & Wang, W. (2006). Sequential pattern mining in multi-databases via multiple alignment. *Data Min. Knowl. Discov., 12*(2-3), 151-180.

Kuo, R. J., Chao, C. M., & Liu, C. Y. (2009). Integration of K-means algorithm and AprioriSome algorithm for fuzzy sequential pattern mining. *Appl. Soft Comput., 9*(1), 85-93.

Laur, P.-A., Symphor, J.-E., Nock, R., & Poncelet, P. (2007). Statistical supports for mining sequential patterns and improving the incremental update process on data streams. *Intell. Data Anal., 11*(1), 29-47.

Lin, N. P., Chen, H.-J., Hao, W.-H., Chueh, H.-E., & Chang, C.-I. (2008). Mining strong positive and negative sequential patterns. *W. Trans. on Comp., 7*(3), 119-124.

Mannila, H., Toivonen, H., & Verkamo, I. (1997). Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery, 1*(3), 259-289.

Masseglia, F., Poncelet, P., & Teisseire, M. (2009). Efficient mining of sequential patterns with time constraints: Reducing the combinations. *Expert Syst. Appl., 36*(3), 2677-2690.

Masseglia, F., Poncelet, P., & Teisseire, M. (2003). Incremental mining of sequential patterns in large databases. *Data Knowl. Eng., 46*(1), 97–121.

Mendes, L. F., Ding, B., & Han, J. (2008). Stream sequential pattern mining with precise error bounds. *Proc. 2008 Int. Conf. on Data Mining (ICDM'08),* Pisa, Italy, Dec. 2008.

Parthasarathy, S., Zaki, M., Ogihara, M., & Dwarkadas, S. (1999). Incremental and interactive sequence mining. *In Proc. of the 8th Int. Conf. on Information and Knowledge Management (CIKM'99).*

Pei, J., Han, J., & Wang, W. (2007). Constraint-based sequential pattern mining: The pattern-growth methods. *J. Intell. Inf. Syst., 28*(2), 133-160.

Pei, J., Han, J., Asl, M. B., Pinto, H., Chen, Q., Dayal, U., et al. (2001). PrefixSpan mining sequential patterns efficiently by prefix projected pattern growth. *Proc.17th Int'l Conf. on Data Eng.*, (pp. 215-226).

Pinto, H., Han, J., Pei, J., Wang, K., Chen, Q., & Dayal, U. (2001). Multi-dimensional sequential pattern mining. *CIKM '01: Proceedings of the Tenth International Conference on Information and Knowledge Management* (pp. 81-88). New York, NY: ACM.

Seno, M., & Karypis, G. (2002). SLPMiner: An algorithm for finding frequent sequential patterns using length-decreasing support constraint. I*n Proceedings of the 2nd IEEE International Conference on Data Mining (ICDM)*, (pp. 418-425).

Srikant, R., & Agrawal, R. (1996). *Advances in database technology EDBT '96*., (pp. 3-17).

Wang, J. L., Chirn, G., Marr, T., Shapiro, B., Shasha, D., & Zhang, K. (1994). Combinatorial pattern discovery for scientific data: Some preliminary results. *Proc. ACM SIGMOD Int'l Conf. Management of Data*, (pp. 115-125).

Xing, Z., Pei, J., & Keogh, E. (2010). A brief survey on sequence classification. *SIGKDD Explorations Newsletter, 12*(1), 40-48.

Yan, X., Han, J., & Afshar, R. (2003). CloSpan: Mining closed sequential patterns in large datasets. *Proceedings of SDM*, (pp. 166-177).

Yun, U. (2008). A new framework for detecting weighted sequential patterns in large sequence databases. *Know.-Based Syst., 21*(2), 110-122.

Zaki, M. J. (2000). Sequence mining in categorical domains: Incorporating constraints. *CIKM '00: Proceedings of the Ninth International Conference on Information and Knowledge Management* (pp. 422-429). New York, NY: ACM.

Zaki, M. J. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, *42*(1-2), 31-60.

Zhang, M., Kao, B., Cheung, D., & Yip, C. (2002). Efficient algorithms for incremental updates of frequent sequences., In *Proc. of the 6th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (PAKDD'02).

ADDITIONAL READING

Adamo, J.-M. (2001). *Data Mining for Association Rules and Sequential Patterns: Sequential and Parallel Algorithms.* Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Alves, R., Rodriguez-Baena, D. S., Aguilar-Ruiz, & S., J. (2009). Gene association analysis: a survey of frequent pattern mining from gene expression data. *Briefings in Bioinformatics* , 210-224.

Fradkin, D., & Moerchen, F. (2010). Margin-closed frequent sequential pattern mining. *UP '10: Proceedings of the ACM SIGKDD Workshop on Useful Patterns* (pp. 45-54). New York, NY, USA: ACM.

Garofalakis, M., Rastogi, R., & Shim, K. (2002). Mining Sequential Patterns with Regular Expression Constraints. *IEEE Trans. on Knowl. and Data Eng.* , 530-552.

Han, J., & Kamber, M. (2006). *Data Mining: Concepts and Techniques, Second edition.* Morgan Kaufmann Publishers.

Joshi, M. V., Karypis, G., & Kumar, V. (2000). Parallel Algorithms for Mining Sequential Associations: Issues and Challenges (2000).

Li, T.-R., Xu, Y., Ruan, D., & Pan, W.-m. Sequential pattern mining. In R. Da, G. Chen, E. E. Kerre, & G. Wets, *Intelligent data mining: techniques and applications* (pp. 103-122). Springer.

Lin, M.-Y., Hsueh, S.-C., & Chan, C.-C. (2009). Incremental Discovery of Sequential Patterns Using a Backward Mining Approach. *Proceedings of the 2009 International Conference on Computational Science and Engineering* (pp. 64-70). Washington, DC, USA: IEEE Computer Society.

Lu, J., Adjei, O., Chen, W., Hussain, F., & Enachescu, C. (n.d.). Sequential Patterns Mining.

Masseglia, F., Cathala, F., & Poncelet, P. *The PSP approach for mining sequential patterns.* Springer.

Shintani, T., & Kitsuregawa, M. (1998). Mining Algorithms for Sequential Patterns in Parallel: Hash Based Approach. *Proceedings of the Second Pacific–Asia Conference on Knowledge Discovery and Data mining*, (pp. 283-294).

Srinivasa, R. N. (2005). Data mining in e-commerce: A survey. *Sadhana* , 275-289.

Teisseire, M., Poncelet, P., Scientifique, P., Besse, G., Masseglia, F., Masseglia, F., et al. (2005). Sequential pattern mining: A survey on issues and approaches. *Encyclopedia of Data Warehousing and Mining, nformation Science Publishing* (pp. 3-29). Oxford University Press.

Tzvetkov, P., Yan, X., & Han, J. (2005). TSP: Mining top-k closed sequential patterns. *KNOWLEDGE AND INFORMATION SYSTEMS* , 438-457.

Wang, W., & Yang, J. (2005). *Mining Sequential Patterns from Large Data Sets (Advances in Database Systems).* Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Yang, L. (2003). Visualizing frequent itemsets, association rules, and sequential patterns in parallel coordinates. *ICCSA'03: Proceedings of the 2003 international conference on Computational science and its applications* (pp. 21-30). Montreal, Canada: Springer-Verlag.

Zhao, Q., & Bhowmick, S. S. (2003). Sequential Pattern Matching: A Survey.