Sequential Pattern Mining – Approaches and Algorithms

CARL H. MOONEY and JOHN F. RODDICK, Flinders University

Sequences of events, items, or tokens occurring in an ordered metric space appear often in data and the requirement to detect and analyze frequent subsequences is a common problem. Sequential Pattern Mining arose as a subfield of data mining to focus on this field. This article surveys the approaches and algorithms proposed to date.

Categories and Subject Descriptors: H.2.8 [**Database Applications**]: Data mining

General Terms: Algorithms, Design

Additional Key Words and Phrases: Sequential pattern mining

ACM Reference Format:
Mooney, C. H. and Roddick, J. F. 2013. Sequential pattern mining – Approaches and algorithms. ACM Comput. Surv. 45, 2, Article 19 (February 2013), 39 pages.
DOI = 10.1145/2431211.2431218. http://doi.acm.org/10.1145/2431211.2431218.

1. INTRODUCTION

1.1. Background and Previous Research

Sequences are common, occurring in any metric space that facilitates either total or partial ordering. Events in time, codons, or nucleotides in an amino acid, Web site traversal, computer networks, and characters in a text string are examples of where the existence of sequences may be significant and where the detection of frequent (totally or partially ordered) subsequences might be useful. Sequential pattern mining has arisen as a technology to discover such subsequences.

The sequential pattern mining problem was first addressed by Agrawal and Srikant [1995] and was defined as follows.

> "Given a database of sequences, where each sequence consists of a list of transactions ordered by transaction time and each transaction is a set of items, sequential pattern mining is to discover all sequential patterns with a user-specified minimum support, where the support of a pattern is the number of data sequences that contain the pattern."

Since then there has been a growing number of researchers in the field, evidenced by the volume of papers produced, and the problem definition has been reformulated in a number of ways. For example, Garofalakis et al. [1999] described it as follows.

> "Given a set of data sequences, the problem is to discover subsequences that are frequent, that is, the percentage of data sequences containing them exceeds a user-specified minimum support"

Authors' addresses: C. H. Mooney (corresponding author), J. F. Roddick, School of Computer Science, Engineering and Mathematics, Flinders University, P.O. Box 2100, Adelaide 5001, South Australia; email: Carl.mooney@flinders.edu.au.

DOI 10.1145/2431211.2431218. http://doi.acm.org/10.1145/2431211.2431218.

And Masseglia et al. [2000] describe it like This.

> "...the discovery of temporal relations between facts embedded in a database"

Zaki [2001b] present it as a process to "...discover a set of attributes, shared across time among a large number of objects in a given database."

Since there are varied forms of dataset (transactional, streams, time series, and so on) algorithmic development in this area, research has largely focused on the development and improvement for specific domain data. In the majority of cases the data has been stored as transactional datasets and similar techniques such as those used by association rule miners [Ceglar and Roddick 2006] have been employed in the discovery process. However, the data used for sequence mining is not limited to data stored in overtly temporal or longitudinally maintained datasets; examples include genome searching, Web logs, alarm data in telecommunications networks, and population health data. In such domains data can be viewed as a series of ordered or semi-ordered events, or episodes, occurring at specific times or in a specific order and therefore the problem becomes a search for collections of events that occur, perhaps according to some pattern, frequently together. Mannila et al. [1997] described the problem as follows.

> "An episode is a collection of events that occur relatively close to each other in a given partial order."

Such episodes can be represented as acyclic digraphs and are thus more general than linearly ordered sequences.

This form of discovery requires a different type of algorithm and will be described separately from those algorithms that are based on the more general transaction-oriented datasets. Regardless of the format of the dataset, sequence mining algorithms can be categorized into one of three broad classes that perform the task [Pei et al. 2002]: Apriori-based, either horizontal or vertical database format, and projection-based pattern growth algorithms. Improvements in algorithms and algorithmic development in general have followed similar developments in the related field of association rule mining and have been motivated by the need to process more data at an increased speed with lower overheads.

As mentioned earlier, much research in sequential pattern mining has been focused on the development of algorithms for specific domains. Such domains include areas such as biotechnology [Wang et al. 2004; Hsu et al. 2007], telecommunications [Ouh et al. 2001; Wu et al. 2001], spatial/geographic domains [Han et al. 1997], retailing/market-basket [Srivastava et al. 2000; Pei et al. 2000a; El-Sayed et al. 2004], and identification of plan failures [Zaki et al. 1998]. This has led to algorithmic developments that directly target real problems and explains, in part, the diversity of approaches, particularly in constraint development, taken in algorithmic development.

Over the past few years, various taxonomies and surveys have been published in sequential pattern mining that also provide useful resources [Zhao and Bhowmick 2003; Han et al. 2007; Pei et al. 2007; Mabroukeh and Ezeife 2010]. In addition, a benchmarking exercise of sequential pattern mining is also available [Kum et al. 2007a]. This survey extends or complements these papers by providing a more complete review including further analysis of the research in areas such as constraints, counting, incremental algorithms, closed frequent patterns, and other areas.

1.2. Problem Statement and Notation

Notwithstanding the preceding definitions, the sequential pattern mining problem might also be stated through a series of examples. For example, consider the following.

—Given a set of alerts and status conditions issued by a system before a failure, can we find sequences or subsequences that might help us to predict failure before it occurs?
—Can we characterize suspicious behavior in a user by analyzing the sequence of commands entered by that user?
—Can we automatically determine the elements of what might be considered "best practice" through analyzing the sequences of actions of experts that lead to good outcomes?
—Is it possible to derive more value from market-basket analysis by including temporal and sequence information?
—Can we characterize users who purchase goods or services from our Web site in terms of the sequence and/or pace at which they browse Web pages? Within this group, can we characterize users who purchase one item against those who purchase multiple items?

Put simply, since many significant events occur over time, space, or some other ordered metric, can we learn more from the data by taking account of any ordered sequences that appear in the data?

More formally, the problem of mining sequential patterns, and its associated notation, can be given as follows.

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, termed *items*, which comprise the alphabet. An *event* is a nonempty unordered collection of items. It is assumed without loss of generality that items of an event are sorted in lexicographic order. A *sequence* is an ordered list of events. An event is denoted as $(i_1, i_2, \ldots, i_k)$, where i_j is an item. A sequence α is denoted as $\langle \alpha_1 \rightarrow \alpha_2 \rightarrow \cdots \rightarrow \alpha_q \rangle$, where α_i is an event. A sequence with k-items, where $k = \sum_j |\alpha_j|$, is termed a *k-sequence*. For example, $\langle B \rightarrow AC \rangle$ is a 3-sequence. A sequence $\langle \alpha_1 \rightarrow \alpha_2 \ldots \rightarrow \alpha_n \rangle$ is a *subsequence* of another sequence $\langle \beta_1 \rightarrow \beta_2 \cdots \rightarrow \beta_m \rangle$ if there exist integers $i_1 < i_2 < \cdots < i_n$ such that $\alpha_1 \subseteq \beta_{i_1}, \alpha_2 \subseteq \beta_{i_2}, \ldots, \alpha_n \subseteq \beta_{i_n}$. For example, the sequence $\langle B \rightarrow AC \rangle$ is a subsequence of $\langle AB \rightarrow E \rightarrow ACD \rangle$, since $B \subseteq AB$ and $AC \subseteq ACD$, and the order of events is preserved. However, the sequence $AB \rightarrow E$ is not a subsequence of ABE and vice versa.

We are given a database $\mathcal{D}$ of input sequences where each input sequence in the database has the following fields: sequence-id, event-time, and the items present in the event. It is assumed that no sequence has more that one event with the same time-stamp, so that the time-stamp may be used as the event identifier. In general, the *support* or *frequency* of a sequence, denoted $\sigma(\alpha, \mathcal{D})$, is defined as the number (or proportion) of input sequences in the database $\mathcal{D}$ that contain α. This general definition has been modified as algorithmic development has progressed and different methods for calculating support have been introduced, a summary of which is included in Section 5.2. Given a user-specified threshold, termed the *minimum support* (denoted *min_supp*), a sequence is said to be *frequent* if it occurs more than *min_supp* times and the set of frequent k-sequences is denoted as $\mathcal{F}_k$. Further, a frequent sequence is deemed to be *maximal* if it is not a subsequence of any other frequent sequence. The task then becomes to find all maximal frequent sequences from $\mathcal{D}$ satisfying a user-supplied support *min_supp*.

This constitutes the problem definition for all sequence mining algorithms whose data are located in a transaction database or are transaction datasets. Further terminology, that is specific to an algorithm or set of algorithms, will be elaborated as required.

1.3. Survey Structure

This article begins with a discussion of the algorithms that have been used in sequence mining. These will be categorized firstly on the type of dataset that each algorithm

Table I. Horizontal Formatting Data Layout – adapted from Agrawal and Srikant [1995]

Customer Transaction Database

Customer Id	Transaction Time	Items Bought
1	June 25 '03	30
1	June 30 '03	90
2	June 10 '03	10, 20
2	June 15 '03	30
2	June 20 '03	40, 60, 70
3	June 25 '03	30, 50, 70
4	June 25 '03	30
4	June 30 '03	40, 70
4	July 25 '03	90
5	June 12 '03	90

Customer-Sequence version

Customer Id	Customer Sequence
1	$\langle$ (30) (90) $\rangle$
2	$\langle$ (10 20) (30) (40 60 70) $\rangle$
3	$\langle$ (30 50 70) $\rangle$
4	$\langle$ (30) (40 70) (90) $\rangle$
5	$\langle$ (90) $\rangle$

accommodates and secondly, within that designation, on the broad class to which they belong. Since time is often an important aspect of a sequence, temporal sequences are then discussed in Section 4. This is followed in Section 5 by a discussion on the nature of constraints used in sequence mining and of counting techniques that are used as measures against user-designated supports. This is then followed by discussions of extensions dealing with closed, approximate, and parallel algorithms and the area of incremental algorithms in Sections 6 and 7. Section 8 discusses some areas of related research and the survey concludes with a discussion of other methods that have been employed and areas of related research.

2. APRIORI-BASED ALGORITHMS

The Apriori family of algorithms has typically been used to discover *intra-transaction* associations and then to generate rules about the discovered associations. However the *sequence* mining task is defined as discovering *inter-transaction* associations—sequential patterns—across the same, or similar data. It is therefore not surprising that the first algorithms to deal with this change in focus were based on the Apriori algorithm [Agrawal and Srikant 1994] using transactional databases as their data source.

2.1. Horizontal Database Format Algorithms

Horizontal formatting means that the original data, Table I(a), is sorted first by Customer Id and then by Transaction Time, which results in a transformed customer sequence database, Table I(b), where the time-stamps from Table I(a) are used to determine the order of events, which is used as the basis for mining. The mining is then carried out using a breadth-first approach.

The first algorithms introduced—AprioriAll, AprioriSome, and DynamicSome—used a 5-stage process [Agrawal and Srikant 1995].

(1) *Sort Phase.* This phase transforms the dataset from the original transaction database (Table I(a)) to a customer sequence database (Table I(b)) by sorting the dataset by customer_id and then by transaction_time.

(2) *Litemset* (large itemset) *Phase.* This phase finds the set of all litemsets L (those that meet minimum support), that is, the set of all large 1-sequences since this is simply $\{\langle l \rangle \mid l \in L\}$. At this stage optimizations for future comparisons can be carried out by mapping the litemsets to a set of contiguous integers.
For example, if the minimum support was given as 25%, using the data from Table I(b), a possible mapping for the large itemsets is depicted in Table II.

(3) *Transformation Phase.* Since there is a need to repeatedly determine which of a given set of long sequences are contained in a customer sequence, each customer sequence is transformed by replacing each transaction with the set of litemsets

Table II. Large Itemsets and a Possible Mapping – Agrawal and Srikant [1995]

Large Itemsets	Mapped To
(30)	1
(40)	2
(70)	3
(40 70)	4
(90)	5

Table III. The Transformed Database including the Mappings – Agrawal and Srikant [1995]

C_Id	Original Customer Sequence	Transformed Customer Sequence	After Mapping
1	⟨(30) (90)⟩	⟨{(30)} {(90)}⟩	⟨{1} {5}⟩
2	⟨(10 20) (30) (40 60 70)⟩	⟨{(30)} {(40), (70), (40 70)}⟩	⟨{1} {2, 3, 4}⟩
3	⟨(30 50 70)⟩	⟨{(30), (70)}⟩	⟨{1, 3}⟩
4	⟨(30) (40 70) (90)⟩	⟨{(30)} {(40), (70), (40 70)} {(90)}⟩	⟨{1} {2, 3, 4} {5}⟩
5	⟨(90)⟩	⟨{(90)}⟩	⟨{5}⟩

contained in that transaction. Transactions that do not contain any litemsets are not retained and a customer sequence that does not contain any litemsets is dropped. The customer sequences that are dropped do, however, still contribute to the total customer count. This is depicted in Table III.

(4) *Sequence Phase.* This phase mines the set of litemsets to discover the frequent subsequences. Three algorithms are presented for this discovery, all of which make multiple passes over the data. Each pass begins with a seed set for producing potential large sequences (candidates) with the support for these candidates calculated during the pass. Those that do not meet the minimum support threshold are pruned and those that remain become the seed set for the next pass. The process begins with the large 1-sequences and terminates when either no candidates are generated or no candidates meet the minimum support criteria.

(5) *Maximal Phase.* This is designed to find all maximal sequences among the set of large sequences. Although this phase is applicable to all of the algorithms, AprioriSome and DynamicSome combine this with the sequence phase to save time by not counting nonmaximal sequences. The process is similar in nature to the process of finding all subsets of a given itemset and as such the algorithm for performing this task is also similar. An example can be found in Agrawal and Srikant [1994].

The difference in the three algorithms results from the methods of counting the sequences produced. Although all are based on the Apriori algorithm [Agrawal and Srikant 1994] the AprioriAll algorithm counts all of the sequences whereas AprioriSome and DynamicSome are designed to only produce maximal sequences and therefore can take advantage of this by first counting longer sequences and only counting shorter ones that are not contained in longer ones. This is done by utilizing a *forward* phase that finds all sequences of a certain length and a *backward* phase that finds the remaining long sequences not discovered during the forward phase.

This seminal work, however, had some limitations.

—Given that the output was the set of maximal frequent sequences, some of the inferences (rules) that could be made could be construed as being of no real value. For example, a retail store would probably not be interested in knowing that a customer purchased product “A” and then some considerable time later purchased product “B”.

—Items were constrained to appear in a single transaction limiting the inferences available and hence the potential value that the discovered sequences could elicit. In many domains it could be beneficial that transactions that occur within a certain time window (the time between the maximum and minimum transaction times) are viewed as a single transaction.

—Many datasets have a user-defined hierarchy associated with them and users may wish to find patterns that exist not only at one level, but across different levels of the hierarchy.

In order to address these shortcomings the Apriori model was extended and resulted in the GSP (Generalized Sequential Patterns) algorithm [Srikant and Agrawal 1996]. The extensions included time constraints (minimum and maximum gap between transactions), sliding windows, and taxonomies. The minimum and/or maximum gap between adjacent elements was included to reduce the number of "trivial" rules that may be produced. For example, a video store owner may not care if customers rented "The Two Towers" a considerable length of time after renting "Fellowship of the Ring", but may if this sequence occurred within a few weeks. The sliding window enhances the timing constraints by allowing elements of a sequential pattern to be present in a set of transactions that occur within the user-specified time window. Finally, the user-defined taxonomy (*is-a* hierarchy) which is present in many datasets allows sequential patterns to include elements from any level in the taxonomy.

The algorithm follows the *candidate generation and prune* paradigm where each subsequent set of candidates is generated from seed sets from the previous frequent pass ($\mathcal{F}_{k-1}$) of the algorithm, where $\mathcal{F}_k$ is the set of frequent k-length sequences. This is accomplished in two steps.

(1) *The join step*. This is done by joining sequences in the following way. A sequence s_1 is joined with s_2 if the subsequence obtained by removing the first item of s_1 is the same as the subsequence obtained by removing the last item of s_2. The candidate sequence is then generated by extending s_1 with the last item in s_2 and the added item becomes a separate element if it was a separate element in s_2 or becomes part of the last element of s_1 otherwise. For example, if $s_1 = \langle(1, 2)\ (3)\rangle$ and for the first case $s_2 = \langle(2)\ (3, 4)\rangle$ and the second case $s_2 = \langle(2)\ (3)\ (4)\rangle$ then after the join the candidate would be $c = \langle(1, 2)\ (3, 4)\rangle$ and $c = \langle(1, 2)\ (3)\ (4)\rangle$ respectively.

(2) *The prune step*. Candidates are deleted if they contain a contiguous $(k-1)$ subsequence whose support count is less than the minimum specified support. A more rigid approach can be taken when there is no max-gap constraint resulting in any subsequence without minimum support being deleted.

The algorithm terminates when there are no frequent sequences from which to generate seeds, or there are no candidates generated.

To reduce the number of candidates that need to be checked, an adapted version of the hash-tree data structure was adopted [Agrawal and Srikant 1994]. The nodes of the hash-tree either contain a list of sequences as a *leaf* node or a hash table as an *interior* node. Each nonempty bucket of a hash table in an interior node points to another node. By utilizing this structure, candidate checking is then performed using either a forward or backward approach. The forward approach deals with successive elements as long as the difference between the end time of the element just found and the start time of the previous element is less than max-gap. If this difference is more than max-gap then the algorithm switches to the backward approach where the algorithm moves backward until either the max-gap constraint between the element just pulled up and the previous element is satisfied or the root is reached. The algorithm then switches to the forward approach and this process continues, switching between the forward and

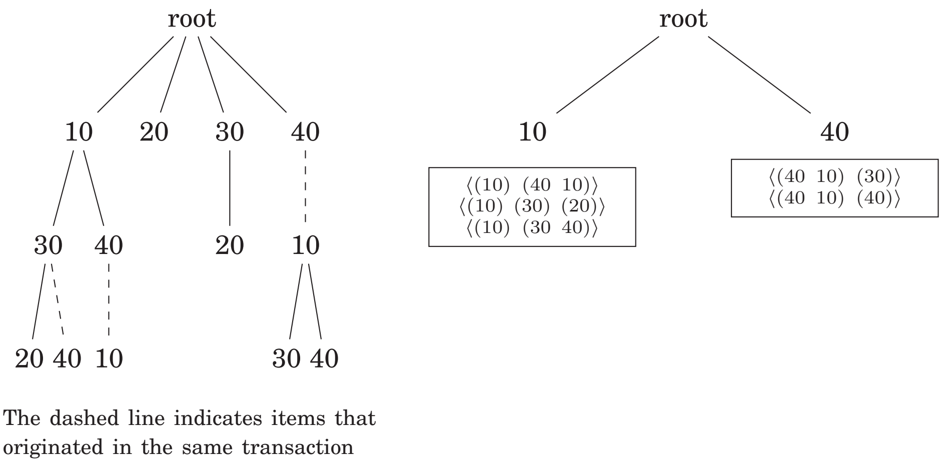


Fig. 1. The prefix-tree of *PSP* (left tree) and the hash-tree of *GSP* (right tree) showing storage after candidate-3 generation [Masseglia et al. 1998].

backward approaches until all elements are found. These improvements in counting and pruning candidates led to improved speed over that of AprioriAll and although the introduction of constraints improved the functionality of the process (from a user perspective) a problem still existed with respect to the number of patterns that were generated. This is an inherent problem facing all forms of pattern mining algorithm with either constraints (see Section 5.1) or approximate patterns (see Section 6.3) being commonly used solutions.

The PSP algorithm [Masseglia et al. 1998] was inspired by GSP, but has improvements that make it possible to perform retrieval optimizations. The process uses transactional databases as its source of data and a candidate generation and scan approach for the discovery of frequent sequences. The difference lies in the way that the candidate sequences are organized. GSP and its predecessors use hash tables at each internal node of the candidate tree, whereas the PSP approach organizes the candidates in a prefix-tree according to their common elements which results in lower memory overhead and faster retrievals. The tree structure used in this algorithm only stores initial subsequences common to several candidates once and the terminal node of any branch stores the support of the sequence to any considered leaf inclusively. Adding to the support value of candidates is performed by navigating to each leaf in the tree and then incrementing the value, which is faster than the GSP approach. A comparison of the tree structures is illustrated in Figure 1 using the following set of frequent 2-sequences: $\mathcal{F}_2 = \langle(10)\,(30)\rangle, \langle(10)\,(40)\rangle, \langle(30)\,(20)\rangle, \langle(30)\,(40)\rangle, \langle(40\ 10)\rangle$. The illustration shows the state of the trees after generating the 3-candidates and shows the reduced overhead of the PSP approach.

To enable users to take advantage of a predefined requirement for output, a family of algorithms termed SPIRIT (Sequential Pattern mIning with Regular expressIon consTraints) was developed [Garofalakis et al. 1999]. The choice of regular expression constraints was due to their expressiveness in allowing families of sequential patterns to be defined and the simple and natural syntax that they provide. Apart from the reduction of potentially useless patterns the algorithms also gained significant performance by "pushing" the constraints inside the mining algorithm, that is using constraint-based pruning followed by support-based pruning and storing the regular expressions in finite state automata. This technique reduces to the candidate generation and pruning of GSP when the constraint, $\mathcal{C}$, is antimonotone, but when this is

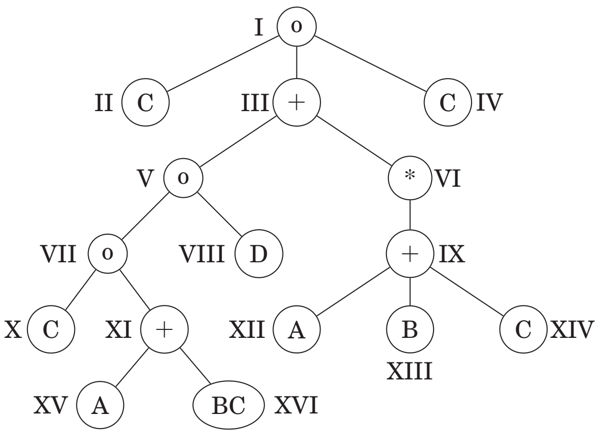


Fig. 2. Hackle-tree for $C((C(A+BC)D)+(A+B+C)^{*})C$ – [Albert-Lorincz and Boulicaut 2003b].

not (as is the case of the regular expressions used by SPIRIT) a relaxation of $\mathcal{C}$, that is a weaker or less restrictive constraint $\mathcal{C}'$, is used. Varying levels of relaxation of $\mathcal{C}$ gave rise to the SPIRIT family of algorithms that are ordered in the following way: SPIRIT(N)("N" for Naive), employs the weakest relaxation, followed by SPIRIT(L)("L" for Legal), SPIRIT(V)("V" for Valid), and SPIRIT(R)("R" for Regular). This decrease in relaxation impacts on the effectiveness of both the constraint-based and support-based pruning but has the potential to increase the performance of the algorithm by restricting the number of candidates that are generated during each pass of the pattern mining loop.

The MFS (Maximal Frequent Sequences) algorithm [Zhang et al. 2001] uses a modified version of the GSP candidate generation function as its core candidate generation function. This modification allows for a significant reduction in any I/O requirements since the algorithm only checks candidates of various lengths in each database scan. The authors call this a *successive refinement* approach. The algorithm first computes a rough estimate of the set of all frequent sequences by using the results of a previous mining run if the database has been updated since the last mining run, or by mining a small sample of the database using GSP. This set of varying length frequent sequences is then used to generate the set of candidates which are checked against the database to determine which are in fact frequent. The *maximal* of these frequent sequences are kept and the process is repeated only on these *maximal* sequences checking any candidates against the database. The process terminates when no more new frequent sequences are discovered in an iteration. A major source of efficiency over GSP is that the supports of longer sequences can be checked earlier in the process.

Using regular expressions and minimum frequency constraints combines both antimonotonic (frequency) and nonantimonotonic (regular expressions) pruning techniques as has already been discussed for the SPIRIT family of algorithms. The RE-Hackle algorithm (Regular Expression-Highly Adaptive Constrained Local Extractor) [Albert-Lorincz and Boulicaut 2003b] uses a hierarchical representation of Regular Expressions which it stores in a Hackle-tree rather than the Finite State Automaton used in the SPIRIT algorithms. They build their RE-constraints for mining using REs built over an alphabet using three operators: union (denoted $+$), concatenation (denoted by $\circ_k$, where k is the overlap deleted from the result), and Kleene closure (denoted *). A Hackle-tree (see Figure 2) is a form of abstract syntax tree that encodes the structure of an RE-constraint and is structured with each inner node containing a operator and the leaves containing atomic sequences. In this manner the tree reflects the way in which the atomic sequences are assembled from the unions, concatenations, and Kleene closures to form the initial RE-constraint.

Once the RE-constraint is constructed as a Hackle-tree, extraction functions are then applied to the nodes of the Hackle-tree and return the candidate sequences that need to be counted with those that are deemed to be frequent used for the next generation. This is done by creating a new extraction phrase and a new Hackle-tree is then built and the process resumes. The process terminates when no more candidates are discovered.

The MSPS (Maximal Sequential Patterns using Sampling) algorithm combines the approach taken in GSP and the supersequence frequency pruning for mining maximal frequent sequences [Luo and Chung 2004]. Supersequence-frequency-based pruning is based on the fact that any subsequence of a frequent sequence is also frequent and therefore can be pruned from the set of candidates. This gives rise to the common bottom-up, breadth-first search strategy that includes, from the second pass over the database, mining a small random sample of the database to generate local maximal frequent sequences. After these have been verified in a top-down fashion against the original database, so that the longest frequent sequences can be collected, the bottom-up search is continued. In addition the authors use a signature technique to overcome any problems that may arise when a set of k-sequences will not fit into memory and candidate generation is required.

The sampling that occurs is related to the work of Toivonen [1996] who used a lowered minimum support when mining the sample resulting in less chance that a frequent itemset is missed. In this work, to make sampling more efficient, a statistical method to adjust the user-specified minimum support is used. The counting approach is similar to the PSP approach in that a prefix tree is used, however, the tree used in this algorithm differs in that it is used to count candidates of different sizes and the size of the tree is considerably smaller because of the supersequence-frequency-based pruning. The tree also has an associated bit vector whose size is that of the number of unique single items in the database where each position is initialized to zero. Setting the bit vector up to assist in counting requires that as each candidate is inserted into the prefix tree its corresponding bit is set to one. As each customer sequence is inserted into the tree it is checked against the bit vector first and those items that do not occur (the bit is set to zero) are trimmed accordingly. Figure 3 illustrates a prefix tree for an incoming customer sequence of $ABCD - ADEFG - B - DH$ trimmed against a bit vector of 10111001. The bit vector is derived as shown because despite the database containing the alphabet $\{A, B, C, D, E, F, G, H\}$ there are no candidates containing B, F, and G and therefore should be ignored during counting. Each node may have two types of children S-extension (the child starts a new itemset) and I-extension (the child is in the same itemset with the item represented by its parent node). S-extensions are represented by a dashed line and I-extensions by a solid line.

2.2. Vertical Database Format Algorithms

Algorithmic development in the sequence mining area, to a large extent, has mirrored that in the association rule mining field [Ceglar and Roddick 2006] in that as improvements in performance were required they resulted from, in the first instance, employing a depth-first approach to the mining, and later by using pattern growth methods (see Section 3). This shift required that the data be organized in an alternate manner, a vertical database format, where the rows of the database consist of object-time-stamped pairs associated with an event. This makes it easy to generate *id-lists* for each event that consist of object-time-stamp rows of events thus enabling all frequent sequences to be enumerated via simple temporal joins of the id-lists. An example of this type of format is shown in Table IV.

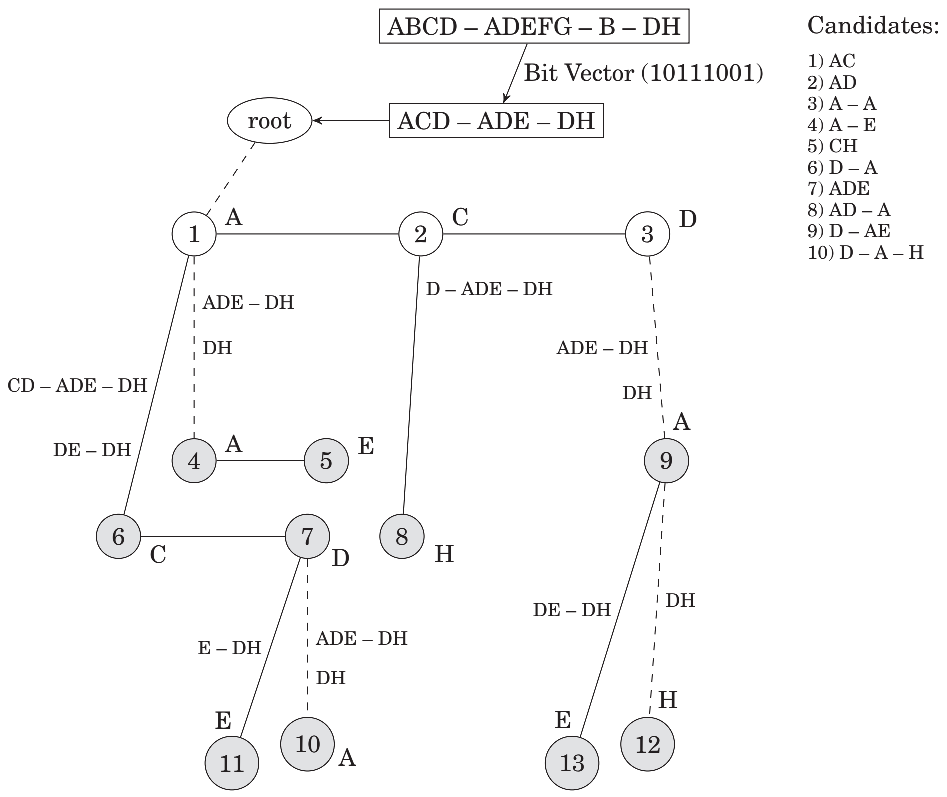


Fig. 3. A prefix tree of MSPS – [Luo and Chung 2004]. The labels on the edges represent recursive calls on segments of the trimmed customer sequence.

Table IV. Vertical Formatting Data Layout – [Zaki 2001b]

(a) Input-Sequence Database

Sequence Id	Time	Items
1	10	C D
1	15	A B C
1	20	A B F
1	25	A C D F
2	15	A B F
2	20	E
3	10	A B F
4	10	D G H
4	20	B F
4	25	A G H

(b) Id-Lists for the Items

A		B		D		F	
SID	EID	SID	EID	SID	EID	SID	EID
1	15	1	15	1	10	1	20
1	20	1	20	1	25	1	25
1	25	2	15	4	10	2	15
2	15	3	10			3	10
3	10	4	20			4	20
4	25						20

SID: Sequence Id
EID: Time

Yang et al. [2002] recognized that these methods perform better when the data is memory resident and when the patterns are long, but also that the generation and counting of candidates becomes easier. This shift in data layout brought about the introduction of algorithms based on a depth-first traversal of the search space and at the same time there was an increased focus on incorporating constraints into the mining process. This is due in part to the improvement in processing time but also as a reaction to the need to reduce the number of results.

The SPADE (Sequential PAttern Discovery using Equivalence classes) algorithm [Zaki 2001b] and its variant cSPADE (constrained SPADE) [Zaki 2000] use

Table V. Computing Support Using Temporal Id-List Joins – adapted from Zaki [2001b]

(a)Intersect D and A, D and B, D and F.

$D \to A$		$D \to B$		$D \to F$	
SID	EID	SID	EID	SID	EID
1	15	1	15	1	20
1	20	1	20	1	25
1	25	4	20	4	20
2	15	3	10		
4	25				

(b)Intersect $D \to B$ and $D \to A$, $D \to B$ and $D \to F$.

$D \to B \to A$		$D \to BF$	
SID	EID	SID	EID
1	20	1	20
1	25	4	20
4	25		

(c)Intersect $D \to B \to A$ and $D \to BF$.

$D \to BF \to A$	
SID	EID
1	25
4	25

combinatorial properties and lattice-based search techniques and allow constraints to be placed on the mined sequences.

The key features of SPADE include the layout of the database in a vertical id-list database format with the search space decomposed into sub-lattices that can be processed independently in main memory thus enabling the database to be scanned only three times or just once on some preprocessed data. Two search strategies are proposed for finding sequences in the lattices.

(1) Breadth-first search: The lattice of equivalence classes is explored in a bottom-up manner and all child classes at each level are processed before moving to the next.
(2) Depth-first search: All equivalence classes for each path are processed before moving to the next path.

Using the vertical id-list database in Table III(b) all frequent 1-sequences can be computed in one database scan. Computing the $\mathcal{F}_2$ can be achieved in one of two ways: by preprocessing and collecting all 2-sequences above a user-specified lower bound, or by performing a vertical to horizontal transformation dynamically.

Once this has been completed the process continues by decomposing the 2-sequences into prefix-based parent equivalence classes followed by the enumeration of all other frequent sequences via either breadth-first or depth-first searches within each equivalence class. The enumeration of the frequent sequences can be performed by joining the id-lists in one of three ways (assume that A and B are items and S is a sequence).

(1) *Itemset and Itemset*: joining AS and BS results in a new itemset ABS.
(2) *Itemset and Sequence*: joining AS with $B \to S$ results in a new sequence $B \to AS$.
(3) *Sequence and Sequence*: joining $A \to S$ with $B \to S$ gives rise to three possible results: a new itemset $AB \to S$, and two new sequences $A \to B \to S$ and $B \to A \to S$. One special case occurs when $A \to S$ is joined with itself resulting in $A \to A \to S$.

The enumeration process is the union or join of a set of sequences or items whose counts are then calculated by performing an intersection of the id-lists of the elements that comprise the newly formed sequence. By proceeding in this manner it is only necessary to use the first two subsequences lexicographically at the last level to compute the support of a sequence at a given level [Zaki 1998]. This process for enumerating and computing the support for the sequence $D \to BF \to A$ is shown in Table V using the data supplied in Table III(b).

The cSPADE algorithm [Zaki 2000] is the same as SPADE except that it incorporates one or more of the following syntactic constraints as checks during the mining process.

(1) Length or width limitations on the sequences allows for highly structured data, for example, DNA sequence databases, to be mined without having the problem of an exponential explosion in the number of discovered frequent sequences.

(2) Minimum or maximum gap constraints on consecutive sequence elements enables the discovery of sequences that occur after a certain minimum amount of time, and no longer than a specified maximum time ahead.
(3) Applying a time window on allowable sequences, requires the entire sequence to occur within the window. This differs from minimum and maximum gaps in that it deals with the entire sequence not the time between sequence elements.
(4) Incorporating item constraints. Since the generation of individual equivalence classes is achieved easily by using the equivalence class approach and a vertical database format, exclusion of an item becomes a simple check for the particular item in the class and the removal from those classes where it occurs. Further expansion of these classes will never contain these items.
(5) Finding sequences predictive of one or more classes (even rare ones). This is only applicable to certain datasets (classification) where each input sequence has a class label.

SPAM (Sequential PAttern Mining using a bitmap representation) [Ayres et al. 2002] uses a novel depth-first traversal of the search space with various pruning mechanisms and a vertical bitmap representation of the database, which enables efficient support counting. A vertical bitmap for each item in the database is constructed while scanning the database for the first time with each bitmap having a bit corresponding to each element of the sequence in the database. One potential limiting factor on its usefulness is its requirement that all of the data fit into main memory.

The candidates are stored in a lexicographic sequence lattice or tree (the same type as used in PSP), which enables the candidates to be extended in one of two ways: *Sequence Extended* using an *S-step* process and *Itemset Extended* using an *I-step* process. This process is the same as the approach taken in GSP [Srikant and Agrawal 1996] and PSP [Masseglia et al. 1998] in that an item becomes a separate element if it was a separate element in the sequence it came from or becomes part of the last element otherwise. These processes are carried out using the bitmaps for the sequences or items in question by *ANDing* them to produce the result. The *S-step* process requires that a *transformed bitmap* first be created by setting all bits less than or equal to the item in question for any transaction to zero and all others to one. This transformed bitmap is then used for ANDing with the item to be appended. Table VI(a) and Table VI(b) illustrate this using the data in Table V(a) and bitmap representation in Table V(b).

The method of pruning candidates is based on downward closure and is conducted on both *S-extension* and *I-extension* candidates of a node in the tree using a depth-first search, which guarantees all nodes are visited. However, if the support for a sequence $s < min\_supp$ at a particular node then no more depth-first search is required on s due to downward closure.

The CCSM (Cache-based Constrained Sequence Miner) algorithm [Orlando et al. 2004] uses a level-wise approach initially but overcomes many problems associated with this type of algorithm. This is achieved by using k-way intersections of *id-lists* to compute the support of candidates (the same as SPADE [Zaki 2001b]) combined with a *cache* that stores intermediate id-lists for future reuse.

The algorithm is similar to GSP [Srikant and Agrawal 1996] because it adopts a level-wise bottom-up approach in visiting the sequential patterns in the tree but it differs since after extracting the frequent $\mathcal{F}_1$ and $\mathcal{F}_2$ from the horizontal database, this pruned database is transformed into a vertical one resulting in the same configuration as SPADE [Zaki 2001b]. The major difference is the use of a *cache* to store intermediate id-lists to speed up support counting. This is achieved in the following way: when a new sequence is generated, and if a common prefix is contained in the cache, then the associated id-list is reused and subsequent lines of the cache are rewritten. This

Table VI. SPAM Data Representation – [Ayres et al. 2002]

(a)Data sorted by CID and TID.

Customer ID (CID)	TID	Itemset
1	1	$\{a, b, c\}$
1	3	$\{b, c, d\}$
1	6	$\{b, c, d\}$
2	2	$\{b\}$
2	4	$\{a, b, c\}$
3	5	$\{a, b\}$
3	7	$\{b, c, d\}$

(b)Bitmap representation of the dataset in (a)

CID	TID	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$
1	1	1	1	0	1
1	3	0	1	1	1
1	6	0	1	1	1
-	-	0	0	0	0
2	2	0	1	0	0
2	2	1	1	1	0
-	-	0	0	0	0
-	-	0	0	0	0
3	5	1	1	0	0
3	7	0	1	1	1
-	-	0	0	0	0
-	-	0	0	0	0

(a)S-Step processing

$(\{a\})$		$(\{a\})_s$		$\{b\}$		$(\{a\}, \{b\})$
1		0		1		0
0		1		1		1
0		1		1		1
0		1		0		0
0	S-step	0		1	result	0
1	$\longrightarrow$	1	&	1	$\longrightarrow$	0
0	process	1		0		0
0		1		0		0
1		0		1		0
0		1		1		1
0		1		0		0
0		1		0		0

(b)I-step processing

$(\{a\}, \{b\})$		$\{d\}$		$(\{a\}, \{b, d\})$
0		1		0
1		1		1
1		1		1
0		0		0
0		0	result	0
0	&	0	$\longrightarrow$	0
0		0		0
0		0		0
0		0		0
1		1		1
0		0		0
0		0		0

enables only a single equality join to be performed between the common prefix and the new item, after which the result of the join is added to cache.

The application goal of IBM (Indexed Bit Map for mining frequent sequences) [Savary and Zeitouni 2005] is to find chains of activities that characterize a group of entities and where the input can be composed of single items. Their approach consists of two phases; first, data encoding and compression, and second, frequent sequence generation that is itself comprised of candidate generation and support checking. Four data structures are used for the encoding and compression as follows: a bit map that is a binary matrix representing the distinct sequences, an SV vector to encode all of the ordered combinations of sequences, an index on the bit map to facilitate direct access to sequences, and an NB table also associated with the bit map to hold the frequencies of the sequences. The algorithm only makes one scan of the database to collect the number of distinct sequences, their frequencies, and the number of sequences by size and in doing so allows for the computing of support for each generated sequence. Candidate generation is conducted in the same manner as GSP [Srikant and Agrawal 1996], PSP [Masseglia et al. 1998], and SPAM [Ayres et al. 2002], except in IBM there is only a need to use the I-extension process since the data has been encoded to single item values. Upon completion of candidate generation, support is determined by first accessing the IBM at the cell where the size of the sequence in question is encoded and then using the SV vector to determine if the candidate is contained in subsequent lines of the IBM. The candidate is then accepted as frequent if the count is larger than a user-specified support. The process terminates under the same conditions as the other algorithms, that is when either no candidates can be generated or there are no frequent sequences obtained.

Table VII. A Summary of Apriori-Based Algorithms

Candidate Generation: Horizontal Database Format			
Apriori (All, Some, Dynamic Some)	[Agrawal and Srikant]	1995	
Generalised Sequential Patterns (GSP)	[Srikant and Agrawal]	1996	Max/Min Gap, Window, Taxonomies
PSP	[Masseglia et al.]	1998	Retrieval optimisations
Sequential Pattern mIning with Regular expressIon consTraints (SPIRIT)	[Garofalakis et al.]	1999	Regular Expressions
Maximal Frequent Sequences (MFS)	[Zhang et al.]	2001	Based on GSP, uses Sampling
Regular Expression-Highly Adaptive Constrained Local Extractor (RE-Hackle)	[Albert-Lorincz and Boulicaut]	2003	Regular Expressions, similar to SPIRIT
Maximal Sequential Patterns using Sampling (MSPS)	[Luo and Chung]	2004	Sampling
Candidate Generation: Vertical Database Format			
Sequential PAttern Discovery using Equivalence classes (SPADE)	[Zaki]	2001	Equivalence Classes
Sequential PAttern Mining (SPAM)	[Ayres et al.]	2002	Bitmap representation
LAst Position INduction (LAPIN)	[Yang and Kitsuregawa]	2004	Uses last position
Cache-based Constrained Sequence Miner (CCSM)	[Orlando et al.]	2004	k-way intersections, cache
Indexed Bit Map (IBM)	[Savary and Zeitouni]	2005	Bit Map, Sequence Vector, Index, NB table
LAst Position INduction Sequential PAttern Mining (LAPIN-SPAM)	[Yang and Kitsuregawa]	2005	Uses SPAM, Uses last position

Yang and Kitsuregawa [2005] base LAPIN-SPAM (Last Position INduction Sequential PAttern Mining) on the same principles as SPAM [Ayres et al. 2002] with the exception of the methods for candidate verification and counting. Where SPAM uses many ANDing operations, LAPIN avoids this through the observation that if the last position of item α is smaller than, or equal to, the position of the last item in a sequence s, then item α cannot be appended to s as a $(k+1)$-length sequence extension in the same sequence [Yang et al. 2005]. This transfers similarly to the I-step extensions. In order to exploit this observation the algorithm maintains an ITEM_IS_EXIST_TABLE in which the last position information is recorded for each specific position and during each iteration of the algorithm there only needs to be a check of this table to ascertain whether the candidate is behind the current position.

3. PATTERN GROWTH ALGORITHMS

It was recognized early that the number of candidates that were generated using Apriori-type algorithms are, in the worst case, exponential. That is, if there was a frequent sequence of 100 elements then $2^{100} \approx 10^{30}$ candidates had to be generated to

Table VIII. A Sequence Database, S, for Use with Examples for FreeSpan and PrefixSpan – [Pei et al. 2001b]

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

find such a sequence. Although methods have been introduced to alleviate this problem, such as constraints, the candidate generation and prune method suffers greatly under circumstances when the datasets are large. Another inherent problem is the repeated scanning of the dataset to check a large set of candidates by some method of pattern matching. The recognition of these problems in the first instance in association mining gave rise to, in that domain, the frequent pattern growth paradigm and the FP-Growth algorithm [Han and Pei 2000]. This was similarly recognized by researchers in the sequence mining domain and algorithms were developed to exploit this methodology.

The frequent pattern growth paradigm removes the need for the candidate generation and prune steps that occur in the Apriori-type algorithms and does so by compressing the database representing the frequent sequences into a frequent pattern tree and then dividing this tree into a set of projected databases, which are mined separately [Han and Pei 2000b].

In comparison to Apriori-type algorithms, pattern growth algorithms, while generally more complex to develop, test, and maintain, can be faster when given large volumes of data. Moreover, the opportunity to undertake multiple scans of stream data are limited and thus single-scan pattern growth algorithms are often the only option.

The sequence database shown in Table VIII will be used as a running example for both FreeSpan and PrefixSpan.

FreeSpan (Frequent pattern-projected Sequential Pattern Mining) [Han et al. 2000a] aims to integrate the mining of frequent sequences with that of frequent patterns and use projected sequence databases to confine the search and growth of the subsequence fragments [Han et al. 2000a].

By using projected sequence databases the method greatly reduces the generation of candidate subsequences. The algorithm firstly generates the frequent 1-sequences, $\mathcal{F}_1$, termed an *f_list*, by scanning the sequence database, and then sorts them into support descending order, for example, from the sequence database in Table VIII, f_list = $\langle a:4, b:4, c:4, d:3, e:3, f:3\rangle$, where $\langle pattern\rangle : count$ is the sequence and its associated frequency. This set can be divided into six subsets: those having item f, those having item e but no f, those having item d but no e or f, and so on until a is reached. This is followed by the construction of a lower-triangular frequent item matrix that is used to generate the $\mathcal{F}_2$ patterns and a set of projected databases. Items that are infrequent such as g are removed and do not take part in the construction of projected databases. The projected databases are then used to generate $\mathcal{F}_3$ and longer sequential patterns. This mining process is outlined next [Han et al. 2000a, Pei et al. 2001a].

To find the sequential patterns containing only item a. This is achieved by scanning the database. In the example, the two patterns that are found containing only item a are $\langle a\rangle$ and $\langle aa\rangle$.

To find the sequential patterns containing the item b but no item after b in f_list. By constructing the $\{b\}$-projected database and for a sequence α in S containing item b a subsequence α' is derived by removing from α all items after b in f_list. Next α' is inserted into the $\{b\}$-projected database resulting in the $\{b\}$-projected database containing the following four sequences: $\langle a(ab)a\rangle$, $\langle aba\rangle$, $\langle (ab)b\rangle$, and $\langle ab\rangle$. By scanning the projected database one more time all frequent patterns containing b but no item after b in f_list are found, which are $\langle b\rangle$, $\langle ab\rangle$,$\langle ba\rangle$, $\langle (ab)\rangle$.

Finding other subsets of sequential patterns. This is achieved by using the same process as outlined before on the $\{c\}$-, $\{d\}$-, ..., $\{f\}$-projected databases.

All of the single projected databases are constructed on the first scan of the original database and the process outlined previously is performed recursively on all projected databases while there are still longer candidate patterns to be mined.

PrefixSpan (Prefix-projected Sequential Pattern mining) [Pei et al. 2001b] builds on the concept of FreeSpan but instead of projecting sequence databases it examines only the prefix subsequences and projects only their corresponding postfix subsequences into projected databases. Using the sequence database, S, in Table VIII with a $min\_supp = 2$ the mining is as follows.

The length-1 sequential patterns are the same as for the f_list in FreeSpan – $\langle a\rangle : 4, \langle b\rangle : 4, \langle c\rangle : 4, \langle d\rangle : 3, \langle e\rangle : 3, \langle f\rangle : 3$. As for FreeSpan, the complete set can be divided into six subsets according to the six prefixes. These are then used to gather those sequences that have them as a prefix. Using the prefix a as an example, when performing this operation FreeSpan considers subsequences prefixed by the first occurrence of a, that is, in the sequence $\langle (ef)(ab)(df)cb\rangle$, only the subsequence $\langle (\_b)(df)cb\rangle$ should be considered, ($(\_b)$ means the last element in the prefix, in this case a, together with b form an element).

The sequences in S containing $\langle a\rangle$ are next projected with respect to $\langle a\rangle$ to form the $\langle a\rangle$-projected database, followed by those with respect to $\langle b\rangle$ and so on. By way of example the $\langle b\rangle$-projected database consists of four postfix sequences: $\langle (\_c)(ac)d(cf)\rangle$, $\langle (\_c)(ae)\rangle$, $\langle (df)cb\rangle$, and $\langle \_c\rangle$. By scanning these projected databases all of the length-2 patterns having the prefix $\langle b\rangle$ can be found. These are $\langle ba\rangle$:2, $\langle bc\rangle$:3, $\langle (bc)\rangle$:3, $\langle bd\rangle$:2, and $\langle bf\rangle$:2. This process is then conducted recursively by partitioning the patterns as before to give those having prefix $\langle ba\rangle$, $\langle bc\rangle$ and so on, and these are mined by constructing projected databases and mining each of them recursively. This is done for each of the remaining single prefixes, $\langle c\rangle$, $\langle d\rangle$, $\langle e\rangle$, and $\langle f\rangle$, respectively.

The major benefit of this approach is that no candidate sequences need to be generated or tested that do not exist in a projected database. That is, PrefixSpan only grows longer sequential patterns from shorter frequent ones, thus making the search space smaller. This results in the major cost being the construction of the projected databases, which can be alleviated by two optimizations. The first, by using a bilevel projection method to reduce the size and number of the projected databases, and second a pseudo-projection method to reduce the cost when a projected database can be wholly contained in main memory.

The SLPMiner algorithm [Seno and Karypis 2002] follows the projection-based approach for generating frequent sequences but uses a *length-decreasing support* constraint for the purpose of finding not only short sequences with high support but also long sequences with a lower support. To this end the authors extended the model they introduced for length-decreasing support in association rule mining [Seno and Karypis 2001, 2005], to use the length of a sequence not an itemset. This is formally defined as follows: Given a sequential database D and a function $f(l)$) that satisfies $1 \geq f(l) \geq f(l+1) \geq 0$ for any positive integer l, a sequence s is frequent if and only if $\sigma_D(s) \geq f(|s|)$.

Under this constraint a sequence can be frequent while its subsequences are infrequent so pruning cannot be performed solely using the downward closure principle and therefore three types of pruning are introduced that are derived from knowledge about the length at which an infrequent sequence becomes frequent. This knowledge about the increase in length is termed the *Smallest Valid Extension*

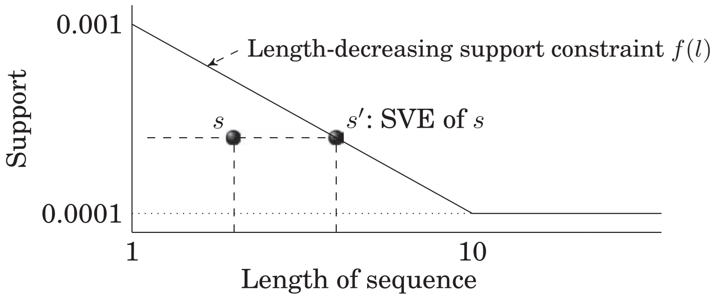


Fig. 4. A typical length-decreasing support constraint and the Smallest Valid Extension (SVE) property – [Seno and Karypis 2002]

Table IX. A Summary of Pattern Growth Algorithms

Algorithm Name	Author	Year	Notes
Pattern Growth			
FREquEnt pattern-projected Sequential PAtterN mining (FreeSpan)	[Han et al.]	2000a	Projected sequence database
PREFIX-projected Sequential PAtterN mining (PrefixSpan)	[Pei et al.]	2001b	Projected prefix database
Sequential pattern mining with Length-decreasing suPport (SLPMiner)	[Seno and Karypis]	2002	Length-decreasing support

property (or *SVE*) and uses the fact that if a line is drawn parallel to the x-axis at $y = \sigma_D(s)$ until it intersects the support curve then the length of the extended sequence is the minimum length the original sequence must attain before it becomes frequent. Figure 4 shows both the length-decreasing support function and the SVE property.

The three methods of pruning are as follows.

(1) *Sequence pruning* removes sequences that occur at every node in the prefix tree from the projected databases. A sequence s can be pruned if the value of the length-decreasing function $f(|s| + |p|)$, where p is the pattern represented at a particular node, is greater than the value of the support for p in the database D.
(2) *Item pruning* removes some of the infrequent items from short sequences. Since the inverse function of the length-decreasing support function yields the length of a particular sequence, an item i can be pruned by determining if the length of a sequence plus the length of the prefix at a node, $|s| + |p|$, is less than the value of the inverse function of the support for the item i in the projected database D' at the current node.
(3) *Min-max pruning* eliminates a complete projected database. This is achieved by splitting the projected database into two subsets and determining if each of them is too small to be able to support any frequent sequential patterns. If this is the case then the entire projected database can be removed.

Although these pruning methods are elaborated for use with SLPMiner the authors note that many of the methods can be incorporated into other algorithms and cite PrefixSpan [Pei et al. 2001b] and SPADE [Zaki 2001b] as two possibilities.

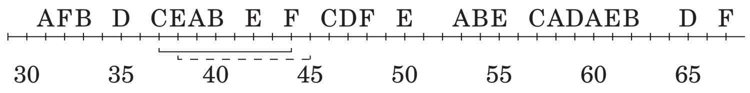


Fig. 5. An example event sequence and two windows of width 7.

4. TEMPORAL SEQUENCES

The data used for sequence mining is not limited to data stored in overtly temporal or longitudinally maintained datasets. In such domains data can be viewed as a series of events occurring at specific times and therefore the problem becomes a search for collections of events that occur frequently together. Solving this problem requires a different approach, and several types of algorithm have been proposed for different domains.

4.1. Problem Statement and Notation for Episode and Event-Based Algorithms

The first algorithmic framework developed to mine datasets that were deemed to be episodic in nature was introduced by Mannila et al. [1995]. The task addressed was to find all episodes that occur frequently in an event sequence, given a class of episodes and an input sequence of events. An episode was defined to be the following.

> "... a collection of events that occur relatively close to each other in a given partial order, and ... frequent episodes as a recurrent combination of events" [Mannila et al. 1995].

The notation used is as follows.

E is a set of event types and an event is a pair (A, t), where $A \in \mathbf{E}$ is an event type and t is an integer (time / occurrence) of the event. There are no restrictions on the number of attributes that an event type may contain, or be made up of, but the original work only considered single values with no loss of generality. An event sequence **s** on **E** is a triple (s, T_s, T_e), where $s = \langle (A_1, t_1), (A_2, t_2), \ldots, (A_n, t_n) \rangle$ is an ordered sequence of events such that $A_i \in \mathbf{E}$ for all $i = 1, \ldots, n$, and $t_i \leq t_{i+1}$ for all $i = 1, \ldots, n-1$. Further $T_s < T_e$ are integers, T_s is termed the starting time and T_e the ending time, and $T_s \leq t_i < T_e$ for all $i = 1, \ldots, n$.

For example, Figure 5 depicts the event sequence $\mathbf{s} = (s, 29, 68)$, where $s = \langle (A, 31), (F, 32), (B, 33), (D, 35), (C, 37), \ldots, (F, 67) \rangle$.

In order for episodes to be considered interesting they must occur close enough in time, which can be defined by the user through a time window of a certain width. These time windows are partially overlapping slices of the event sequence and the number of windows that an episode must occur in to be considered frequent, *min_freq*, is also defined by the user. Notationally a window on event sequence $\mathcal{S} = (s, T_s, T_e)$ is an event sequence $\mathbf{w} = (w, t_s, t_e)$ where $t_s < T_e$, $t_e > T_s$ and w consists of those pairs (A, t) from s where $t_s \leq t < t_e$. The time span $t_e - t_s$ is termed the width of the window **w** denoted $width(\mathbf{w})$. Given an event sequence **s** and an integer *win*, the set of all windows **w** on **s** such that $width(\mathbf{w}) = win$ is denoted by $\mathcal{W}(\mathbf{s}, win)$. Given the event sequence $\mathbf{s} = (s, T_s, T_e)$ and window width *win*, the number of windows in $\mathcal{W}(\mathbf{s}, win)$ is $T_e - T_s + win - 1$.

An *episode* $\varphi = (V, \leq, g)$ is a set of nodes V, a partial order $\leq$ on V, and a mapping $g : V \rightarrow E$ associating each node with an event type. The interpretation of an episode is that it has to occur in the order described by $\leq$.

There are two types of episodes considered.

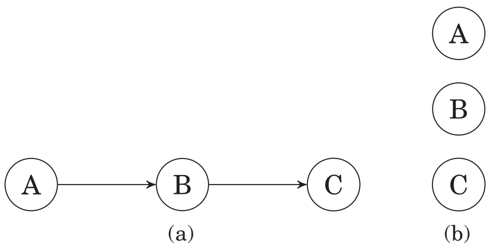


Fig. 6. Depiction of a serial and parallel episode [Mannila et al. 1995].

Serial. This is where the partial order relation $\leq$ is a total order resulting in, for example, an event A preceding an event B which precedes event C. This is shown in Figure 6(a).

Parallel. This is where there are no constraints on the relative order of events, that is if the partial order relation $\leq$ is trivial: ($x \nleq y \ \forall\ x \neq y$). This is shown in Figure 6(b).

The *frequency* of an episode is defined to be the number of windows in which the episode occurs. That is, given an event sequence $\mathbf{s}$ and a window width *win*, the frequency of an episode φ in $\mathbf{s}$ is

$$fr(\varphi, \mathbf{s}, win) = \frac{|\{\mathbf{w} \in \mathcal{W}(\mathbf{s}, win) | \varphi \ occurs\ in\ \mathbf{w}\}|}{|\mathcal{W}(\mathbf{s}, win)|}.$$

Thus, given a *frequency threshold min_freq*, φ is frequent if $fr(\varphi, \mathbf{s}, win) \geq min\_freq$. The task is to discover all frequent episodes (serial or parallel) from a given class $\mathcal{E}$ of episodes.

The goal of WINEPI [Mannila et al. 1995] was given an event sequence $\mathbf{s}$, a set $\mathcal{E}$ of episodes, a window width *win*, and a frequency threshold *min_freq*, to find the set of frequent episodes $\mathcal{F}$ with respect to $\mathbf{s}$, *win* and *min_freq* denoted as $\mathcal{F}(\mathbf{s}, win, min\_freq)$. The algorithm follows a traditional level-wise (breadth-first) search starting with the general episodes (one event). At each subsequent level the algorithm first computes a collection of candidate episodes, checks their frequency against *min_freq* and, if greater, the episode is added to a list of frequent episodes. This cycle continues until there are no more candidates generated or no candidates meet the minimum frequency. This is a typical Apriori-like algorithm under which the downward closure principal holds; if α is frequent then all subepisodes $\beta \preceq \alpha$ are frequent.

In order to recognize episodes in sequences two methods are necessary, one for parallel episodes and one for serial episodes. However, since both of these methods share a similar feature, namely that two adjacent windows are typically similar to each other, the recognition can be done incrementally. For parallel episodes a count is maintained that indicates how many events are present in any particular window and when this count reaches the length of episode (at any given iteration of the algorithm) the index of the window is saved. When the count decreases, indicating that the episode is not entirely in the window, the occurrence field is incremented by the number of windows in which the episode was fully contained. Serial episodes are recognized using state automata that accept the candidate episodes and ignore all other input [Mannila et al. 1995]. A new instance of the automata is initialized for each serial episode every time the first event appears in the window, and the automata is removed when this same event leaves the window. To count the number of occurrences, an automata is said to be in the accepting state when the entire episode is in the window and each time this occurs the automata is removed and its window index is saved. When there are no automata left for the particular episode the occurrence is incremented by the number of saved indexes.

Mannila and Toivonen [1996] extended this work by considering only minimal occurrences of episodes, which are defined as follows. Given an episode φ and an event sequence $\mathbf{s}$, then the interval $[t_s, t_e)$ is a *minimal occurrence* of φ in $\mathbf{s}$ if:

(1) φ occurs in the window $\mathbf{w} = (w, t_s, t_e)$ on $\mathbf{s}$, and
(2) φ does not occur in any proper subwindow on $\mathbf{w}$, that is, $\nexists \mathbf{w}' = (w', t'_s, t'_e)$ on $\mathbf{s}$ such that $t_s \leq t'_s, t'_e \leq t_e$ and $width(\mathbf{w}') < width(\mathbf{w})$.

The algorithm MINEPI takes advantage of this new formalism and follows the same basic principles as WINEPI with the exception that minimal occurrences of candidate episodes are located during the candidate generation phase of the algorithm. This is performed by selecting from a candidate episode φ two subepisodes φ_1 and φ_2 such that φ_1 contains all events of φ except the last one and φ_2 contains all events except the first one. From these two subepisodes the minimal occurrences of φ are found using the following specification [Mannila and Toivonen 1996].

$$\begin{aligned} mo(\varphi) = \{[t_s, u_e) \mid \exists\ [t_s, t_e) \in mo(\varphi_1) \wedge [u_s, u_e) \in mo(\varphi_2) \\ \text{such that } t_s < u_s, t_e < u_e \text{ and } [t_s, u_e) \text{ is minimal}\} \end{aligned}$$

These minimal occurrences are then used to obtain statistical confidences of the episode rules without the need to rescan the data.

Typically, WINEPI produces rules that are similar to association rules. For example, if **ABC** is a frequent sequence then **AB** $\Rightarrow$ **C** (confidence γ) states that if **A** occurs followed by **B** then some time later **C** occurs. The MINEPI and its new formalism of episodes allows for more useful rule formulations, for example: “Dept Home Page”, “Sem. I 2005” [15s] $\Rightarrow$ “Classes in Sem. I 2005” [30s] (γ 0.83). This is read as if a person navigated to the Department Home Page followed by the Semester I page within 15 seconds then within the next 30 seconds she would navigate to the Semester I Classes page 83% of the time.

Huang et al. [2004] introduced the algorithm PROWL (Projected Window Lists) to mine inter-transaction rules and the concept of *frequent continuities* to distinguish their rules from those of intra-transaction or association rules. They also introduced a *don’t care* symbol into the sequence to allow for partial periodicity. PROWL is a two-phase algorithm for mining such frequent continuities and utilizes a projected window list and a depth-first enumeration of the search space.

The definition of a sequence follows the pattern from Section 4.1, however, the authors do not define the complete event sequence as a triple but as a tuple of the form (tid, x_i) where tid is a time instant and x_i is an event. The *continuity pattern* is defined to be a nonempty sequence with window W, $P = (p_1, p_2, \dots, p_w)$ where p_1 is an event and the remainder can either be events or the $*$ (*don’t care*) token. A continuity pattern is termed an *i-continuity* or has length i if exactly i positions in P contain an event. For example, $\{A,^*,^*\}$ is a 1-continuity and $\{A,^*, C\}$ is a 2-continuity. They define the problem to be the discovery of all patterns P with a window W where any subsequence of W in S supports P.

The algorithm uses a Projected Window List (PWL) to grow the sequences where a PWL is defined as $P = \{e_1, e_2, \dots, e_k\}$ and $P.PWL = \{w_1, w_2, \dots, w_k\}$, $w_i = e_i + 1$ for $1 \leq i \leq k$. By concatenating each event with the events in the PWL, longer sequences can be generated for which the number of events in the concatenated list can be checked against the given support and accepted as frequent if it equals or exceeds the value. This process is applied recursively until the projected window lists become empty or the window of a continuity is greater than the maximum window. Table X shows the

Table X. Vertical Layout of the Event Sequences for the PROWL Algorithm [Huang et al. 2004]

Event	Time List	Projected Window List
A	1, 4, 7, 8, 11, 14	2, 5, 8, 9, 12, 15
B	3, 6, 9, 12, 16	4, 7, 10, 13
C	2, 10, 15	3, 11, 16
D	5, 13	6, 14

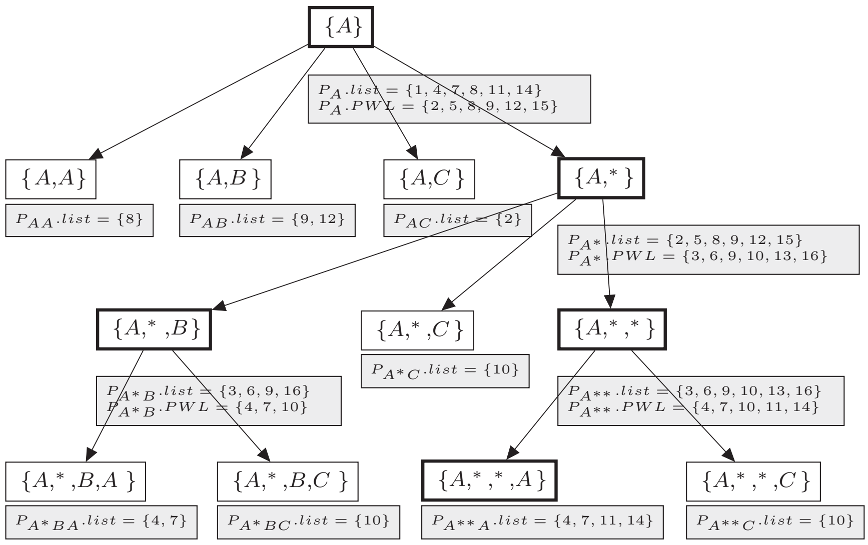

{A}
P_A.list = {1, 4, 7, 8, 11, 14}
P_A.PWL = {2, 5, 8, 9, 12, 15}
{A,A}
P_AA.list = {8}
{A,B}
P_AB.list = {9, 12}
{A,C}
P_AC.list = {2}
{A,*}
P_A*.list = {2, 5, 8, 9, 12, 15}
P_A*.PWL = {3, 6, 9, 10, 13, 16}
{A,*,B}
P_A*B.list = {3, 6, 9, 16}
P_A*B.PWL = {4, 7, 10}
{A,*,C}
P_A*C.list = {10}
{A,*,*}
P_A**.list = {3, 6, 9, 10, 13, 16}
P_A**.PWL = {4, 7, 10, 11, 14}
{A,*,B,A}
P_A*BA.list = {4, 7}
{A,*,B,C}
P_A*BC.list = {10}
{A,*,*,A}
P_A**A.list = {4, 7, 11, 14}
{A,*,*,C}
P_A**C.list = {10}

Fig. 7. The recursive process for the continuity pattern {A} [Huang et al. 2004].

vertical layout of the event sequences and Figure 7 shows the recursive process for the continuity pattern {A}.

Support and confidence are defined in a similar fashion to association rule mining and therefore rules can be formed which are an implication of the form $X \Rightarrow Y$, where X and Y are continuity patterns with window w_1 and w_2 respectively and the concatenation $X \cdot Y$ is a continuity pattern with window $w_1 + w_2$. This leads to support being equal to the support of the concatenation divided by the number of transactions in the event sequence, and confidence as the support of the concatenation divided by the support of either continuity depending on the required implication.

4.2. Event-Oriented Patterns

Sun et al. [2003] approach the problem of mining sequential patterns as that of mining temporal event sequences that lead to a specific *target event*, rather than finding all frequent patterns. They discuss two types of patterns; an *Existence pattern* α with a temporal constraint T that is a set of event values and a *Sequential pattern* β also with a temporal constraint T. Their method is used to produce rules of the type $r = \left\{LHS \xrightarrow{T} e\right\}$, where e is a *target event* value and LHS is a pattern of type α or β. T is a time interval that specifies both the temporal relationship between LHS and e and also the temporal constraint pattern of LHS. To find the LHS patterns they first locate all of the *target events* in the event sequence and create a time-stamp set by using a T-sized window extending back from the target event. The sequence fragments

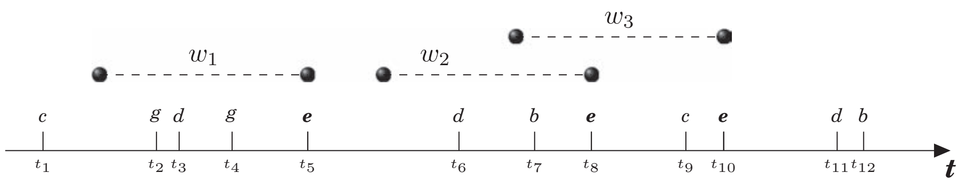

w_1
w_2
w_3
c g d g e d b e c e d b
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12}
t

Fig. 8. Sequence fragments of size T ($w_1 = \langle(g, t_2), (d, t_3), (g, t_4)\rangle$, $w_2 = \langle(d, t_6), (b, t_7)\rangle$, $w_3 = \langle(b, t_7), (e, t_8), (c, t_9)\rangle$) for the target event $\boldsymbol{e}$ in an event sequence s [Sun et al. 2003].

(f_i) that are created from this process are termed the *dataset* of target event e (see Figure 8 for an example).

The support for a rule is then given by the number of sequence fragments containing a specified pattern divided by the total number of sequence fragments in the event sequence. As the authors point out, this method finds frequent sequences that occur not only before target events but elsewhere in the event sequence and therefore it cannot be concluded that these patterns relate to the given target events. In order to prune these nonrelated patterns a confidence measure is introduced which evaluates the number of times the pattern actually leads to the target event divided by the total number of times the pattern occurs. Both of these values are defined as window sets. The formal definition of the problem is then: Given a sequence s, target event value e, window size T, two thresholds s_0 and c_0, find the complete set of rule $r = \{LHS \overset{T}{\rightarrow} e\}$ such that $supp(r) \geq s_0$ and $con(r) \geq c_0$.

4.3. Pattern Directed Mining

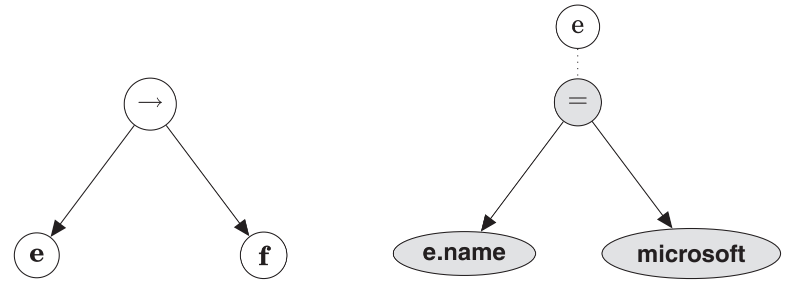
Guralnik et al. [1998] present a framework for the mining of frequent episodes using a pattern language for specifying episodes of interest. A *sequential pattern tree* is used to store the relationships specified by the pattern language and a standard bottom-up mining algorithm can then be used to generate the frequent episodes. The specification for mining follows the notation described earlier (see Section 4.1) with the addition of a *selection constraint* on an event, which is a unary predicate $\alpha(e, a_i)$ on a domain D_i where a_i is an attribute of e, and a *join constraint* on events e and f, which is a binary predicate $\beta(e.a_i,\ f.a_j)$ on a domain $D_i \times D_j$ where a_i and a_j are attributes of e and f respectively. They also define a *sequential pattern* as a combination of partially ordered event specifications constructed from both selection and join constraints. To facilitate the mining process the user-specified patterns are stored in a *Sequential Pattern Tree* (*SP Tree*) where the leaf nodes represent events and the interior nodes represent ordering constraints. In addition, each node holds events matching constraints of that node and attached to the node is a boolean expression that represents the attribute constraints associated with the node (see Figure 9 for two examples).

The mining algorithm constructs the frequent episodes in a bottom-up fashion by taking an SP Tree T and a sequence of events S and at the leaf level matching events against any selection constraints and pruning out those that do not match. The interior nodes merge events of left and right children according to any ordering and join constraints, again pruning out those that do not match the node specifications. The process is continued recursively until all events in S have been visited.

5. CONSTRAINTS AND COUNTING

5.1. Types of Constraints

Constraints to guide the mining process have been employed by many algorithms, not only in sequence mining but also in association mining [Fu and Han 1995; Ng et al.



(a) SP Tree for the user specified pattern $e \rightarrow f$.

(b) SP Tree for the user specified pattern $e\{e.name = \text{'}microsoft\text{'}\}$.

Fig. 9. Two examples of SP Trees [Guralnik et al. 1998].

Table XI. A Summary of Temporal Sequence Algorithms

Algorithm Name	Author	Year	Notes
Candidate Generation: Episodes			
WINEPI	[Mannila et al.]	[1995], 1996	State automata, Window
WINEPI, MINEPI	[Mannila et al.]	1997	State automata, Window, Maximal
Pattern Directed Mining	[Guralnik et al.]	1998	Pattern language
Event-Oriented Patterns	[Sun et al.]	2003	Target events
Pattern Growth: Episodes			
PROjected Window Lists (PROWL)	[Huang et al.]	2004	Projected window lists

1998; Chakrabarti et al. 1998; Bayardo and Agrawal 1999; Pei et al. 2001a; Ceglar et al. 2003]. In general constraints that are imposed on sequential pattern mining, regardless of the algorithms employed, can be categorized into one of eight main types[1]. Other types of constraints will be discussed as they arise, however, in general a constraint C on a sequential pattern α is a boolean function of the form $C(\alpha)$. For the constraints listed shortly only the *duration* and *gap* constraints rely on a support threshold (*min_supp*) to confine the mined patterns; for the others, whether or not a given pattern satisfies a constraint can be determined by the pattern itself. These constraints rely on a set of definitions that are outlined next.

Let $\mathcal{I} = \{x_1, \ldots, x_n\}$ be a set of *items*, each possibly being associated with a set of *attributes*, such as value, price, period, and so on. The value of attribute A of item x is denoted as $x.A$. An *itemset* is a nonempty subset of items, and an itemset with k items is termed a *k-itemset*. A *sequence* $\alpha = \langle X_1 \ldots X_l \rangle$ is an ordered list of itemsets. An itemset $X_i, (1 \le i \le l)$ in a sequence is termed a *transaction*. A transaction X_i may have a special attribute, *time-stamp*, denoted as $X_i.time$, that registers the time when the transaction was executed. The number of transactions in a sequence is termed the *length* of the sequence. A sequence α with length l is termed an *l-sequence*, denoted as len$(\alpha) = l$, with the i-th item denoted as α[i]. A sequence $\alpha = \langle X_1 \ldots X_n \rangle$ is termed a *subsequence*

[1]This is not a complete list but categorizes those that are of most interest and are currently in use. Many of these were discussed first by Pei et al. [2002].

of another sequence $\beta = \langle Y_1 \dots Y_m \rangle$ $(n \leq m)$, and β a *supersequence* of α, denoted as $\alpha \sqsubseteq \beta$, if there exist integers $1 \leq i_1 < \cdots < i_n \leq m$ such that $X_1 \subseteq Y_{i_1}, \dots, X_n \subseteq Y_{i_n}$. A *sequence database SDB* is a set of 2-tuples $(sid,\ \alpha)$, where *sid* is a *sequence-id* and α a sequence. A tuple $(sid,\ \alpha)$ in a sequence database *SDB* is said to contain a sequence γ if γ is a subsequence of α. The number of tuples in a sequence database *SDB* containing sequence γ is termed the support of γ, denoted as $sup(\gamma)$.

The categories of constraints given next appear throughout (expanded from Pei et al. [2002]).

(1) *Item Constraint*. This type of constraint indicates which type of items, singular or groups, are to be included or removed from the mined patterns. The form is $C_{item}(\alpha) \equiv (\varphi i : 1 \leq i \leq len(\alpha),\ \alpha[i]\ \theta\ V)$, or $C_{item}(\alpha) \equiv (\varphi i : 1 \leq i \leq len(\alpha),\ \alpha[i] \cap V \neq \emptyset)$, where V is a subset of items, $\varphi \in \{\forall, \exists\}$ and $\theta \in \{\subseteq, \not\subseteq, \supseteq, \not\supseteq, \in, \notin\}$.
For example, when mining sequential patterns from medical data a user may only be interested in patterns that have a reference to a particular hospital. If H is the set of hospitals, the corresponding item constraint is $C_{hospital}(\alpha) \equiv (\exists i : 1 \leq i \leq len(\alpha), \alpha[i] \subseteq H)$.

(2) *Length Constraint*. This type of constraint specifies the length of the patterns to be mined either as the number of elements or the number of transactions that comprises the pattern. This basic definition may also be modified to include only unique or distinct items.
For example, a user may only be interested in discovering patterns of at least 20 elements in length that occur in the analysis of a sequence of Web log clicks. This might be expressed as a length constraint $C_{len}(\alpha) \equiv (len(\alpha) \leq 20)$.

(3) *Model-Based Constraint*. These types of constraints are those that look for patterns that are either *subpatterns* or *superpatterns* of some given pattern. A *superpattern constraint* is in the form of $C_{pat}(\alpha) \equiv (\exists \gamma \in P \wedge \gamma \sqsubseteq \alpha)$, where P is a given set of patterns. Simply stated this says find patterns that contain a particular set of patterns as subpatterns.
For example, an electronics shop may employ an analyst to mine its transaction database and in doing so may be interested in all patterns that contain first buying a PC then a digital camera and a photo-printer together. This can be expressed as $C_{pat}(\alpha) \equiv \langle (PC)(digital\_camera,\ photo\text{-}printer) \rangle \sqsubseteq \alpha$.

(4) *Aggregate Constraint*. These constraints are imposed on an aggregate of items in a pattern, where the aggregate function can be *avg*, *min*, *max*, etc.
For example, the analyst in the electronics shop may also only be interested in patterns where the average price of all the items is greater than $500.

(5) *Regular Expression Constraint*. This is specified using the established set of regular expression operators, such as disjunction, Kleene closure, etc. For a sequential pattern to satisfy a particular regular expression constraint, C_{RE}, it must be accepted by its equivalent deterministic finite automata, for example, to discover sequential patterns about a patient who was admitted with measles and was given a particular treatment, a regular expression of the form *Admitted(Measles | German Measles)(Treatment A | Treatment B | Treatment C)* where "|" indicates disjunction.

(6) *Duration Constraint*. For these constraints to be defined, every transaction in the sequence database or each item in a temporal sequence must have an associated time-stamp. The duration is determined by the time difference between the first and last transaction, or first and last item, in the pattern and can be either longer or shorter than a given period. Formally, a duration constraint is in the form of $Dur(\alpha)\theta\ \Delta t$, where $\theta \in \{\leq, \geq\}$ and Δt is a value indicating the temporal duration. A sequence α satisfies the constraint (checking satisfaction is achieved by examining

the *SDB*) if and only if $|\{\beta \in SDB | \exists 1 \leq i_1 < \cdots < i_{len(\alpha)} \leq len(\beta)$ such that $\alpha[1] \sqsubseteq \beta[i_1], \ldots, \alpha[len(\alpha)] \sqsubseteq \beta[i_{len(\alpha)}]$, and $(\beta[i_{len(\alpha)}].time - \beta[i_1].time)\theta\ \Delta t\}| \geq min\_supp$. In algorithms that have temporal sequences as their input this type of constraint is usually implemented as a "sliding window" across the event set.

(7) *Gap Constraint*. The patterns that are discovered using this type of constraint are similarly derived from sequence databases or temporal sequences that include time-stamps. The gap is determined by the time difference between adjacent items or transactions and must be longer or shorter than a given gap. Formally, a gap constraint is in the form of $Gap(\alpha)\theta\ \Delta t$, where $\theta \in \{\leq, \geq\}$ and Δt is a value indicating the size of the gap. A sequence α satisfies the constraint (to check the satisfaction of this constraint the *SDB* must be examined) if and only if $|\{\beta \in SDB | \exists 1 \leq i_1 < \cdots < i_{len(\alpha)} \leq len(\beta)$ such that $\alpha[1] \sqsubseteq \beta[i_1], \ldots, \alpha[len(\alpha)] \sqsubseteq \beta[i_{len(\alpha)}]$, and for all $1 < j \leq len(\alpha)$, $(\beta[i_j].time - \beta[i_{j-1}].time)\theta\ \Delta t\}| \geq min\_supp$.

(8) *Timing Marks*. Some tokens in a string may have links to absolute time such as being tokens representing clock ticks. As a result, sequences can be constrained to be not only on time-stamps (such as those discussed in the two cases before) but also in terms of the maximum (or minimum) number of specific timing marks (as special tokens). In this case, the duration and gap constraints can be modified to be of the form $Dur_{TM}(\alpha)\theta\ \tau m$ or $Gap_{TM}(\alpha)\theta\ \tau m$, where $\theta \in \{\leq, \geq\}$ and τm is a value indicating the number of timing marks allowed/required within the sequence [Mooney and Roddick 2006].

5.2. Counting Techniques

For a sequence to be deemed interesting it is required to meet specific criteria in relation to the number of occurrences of it in the sequence with respect to any timing constraints that may have been imposed. In the previous section only two constraints deal with timing (*duration* and *gap*) so for the purpose of this discussion the following terminology will be used (modified from Joshi et al. [1999]).

—*ms*: *maximum span*. This is the maximum allowed time difference between the latest and earliest occurrences of events in the *entire* sequence.

—*ws*: *event-set window size*. This is the maximum allowed time difference between the latest and earliest occurrences of events in any *event-set*. Here the term *event-set* is equivalent to the term *transaction* in Section 5.1.

—*xg*: *maximum gap*. This is the maximum allowed time difference between the latest occurrence of an event in an event set and the earliest occurrence of an event in its immediately preceding event set.

—*ng*: *minimum gap*. This is the minimum required time difference between the earliest occurrence of an event in an event set and the latest occurrence of an event in its immediately preceding event set.

These four parameters can be used to define five different counting techniques that can be divided into three groups. The first group CEVT (count event) searches for the given sequence in the entire sequence timeline. The second group deals with counting the number of windows (windows equate to the maximum span (*ms*)) in which the given sequence occurs and consists of CWIN (count windows) and CMINWIN (count minimum windows). The third group deals with distinct events within the window that is of size *ms* (maximum span) and comprises CDIST (count distinct) and CDIST_O (count distinct with the possibility of overlap). The use of any of these methods depends on the specific application area and on the user's expertise in the domain of the area being mined. Figure 10 shows the differences between the counting methods, and highlights the need for careful consideration when choosing a counting method.

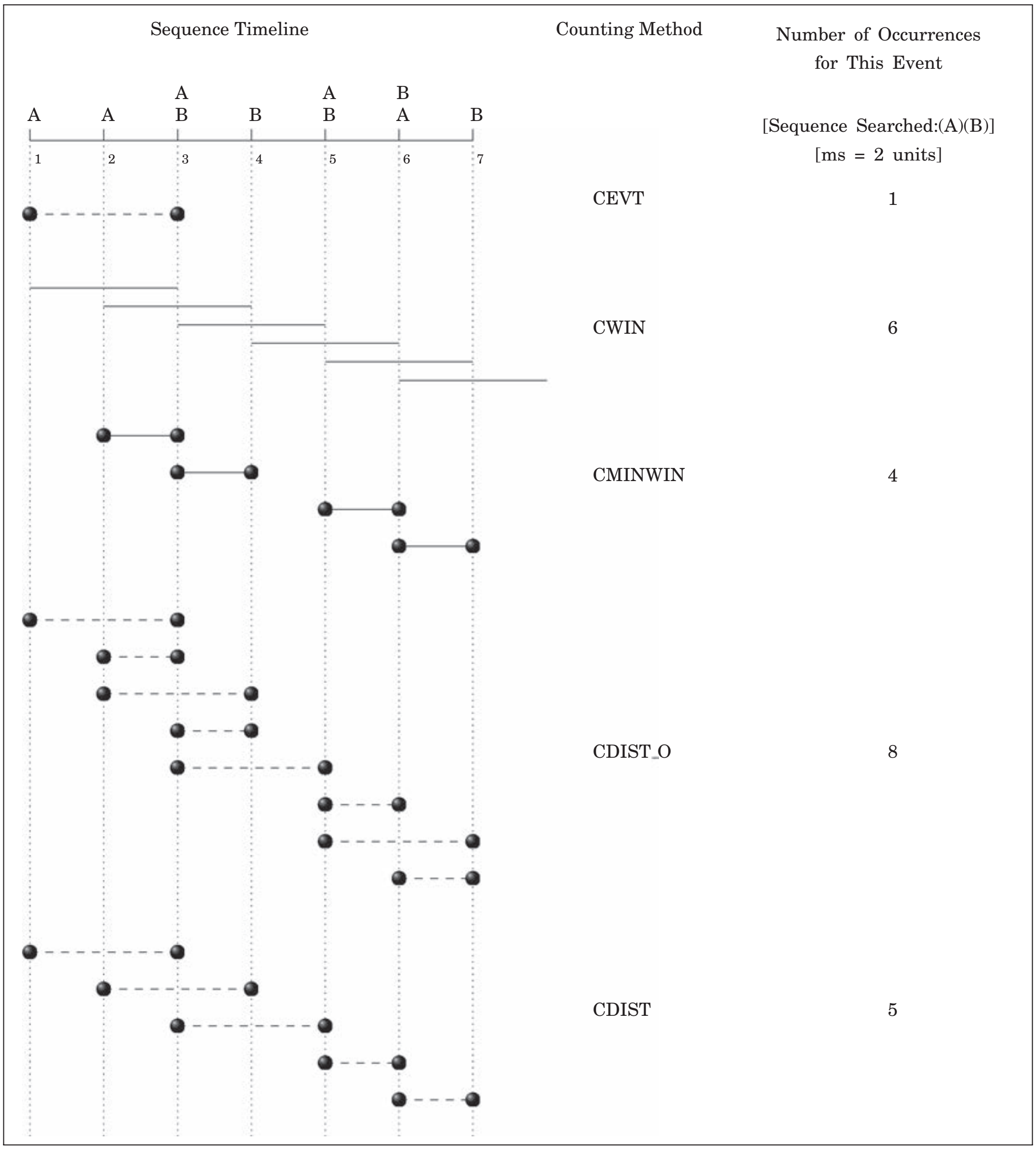


Fig. 10. A comparison of different counting methods Joshi et al. [1999].

6. EXTENSIONS

6.1. Closed Frequent Patterns

The mining of frequent patterns for both sequences and itemsets has proved valuable but in some cases, particularly when using candidate generation and test techniques, and when low support is used, the performance of algorithms can degrade dramatically. This has led to algorithms such as CHARM [Zaki and Hsiao 2002], CLOSET [Pei et al. 2000], CLOSET+ [Wang et al. 2003], CARPENTER [Pan et al. 2003], and Frecpo [Pei et al. 2006], that produce *frequent closed itemsets*, which overcome some of these difficulties and produce a smaller yet complete set of results. In the sequence mining field there are few algorithms that deal with this problem, however, those that do are generally extensions of algorithms that mine the complete set of sequences with improvements in search space pruning and traversal techniques.

A closed subsequence is a sequence that contains no supersequence with the same support. Formally this is defined as follows: given a set of *frequent sequences*, FS, which includes all of the sequences whose support is greater than or equal to *min_supp*, then the set of *closed sequences*, *CS*, is $CS = \{\varphi | \varphi \in FS \text{ and } \nexists \gamma \in FS \text{ such that } \varphi \sqsubseteq \gamma \text{ and } support(\varphi) = support(\gamma)\}$. This results in $CS \subseteq FS$ and the problem of *closed sequence mining* becomes the search for *CS* above a specified support.

Closed *S*equential patter*n* mining [Yan et al. 2003] follows the candidate generation and test paradigm and stores the generated sets of candidates in a hash-indexed result-tree following which postpruning is conducted to produce the closed set of frequent sequences. The algorithm uses a lexicographic sequence tree to store the generated sequences using both I-extension and S-extension mechanisms and the same search tree structure as PrefixSpan to discover all of the frequent sequences (closed and nonclosed). Pruning is conducted using *early termination by equivalence*, a *backward subpattern*, and, *backward superpattern* methods.

BI-Directional Extension-based frequent closed sequence mining mines frequent closed sequences without the need for candidate maintenance [Wang and Han 2004]. Furthermore pruning is made more efficient by adopting a *BackScan* method and the *ScanSkip* optimization technique. The mining is performed using a method termed *BI-Directional Extension* where the forward directional extension is used to grow all of the prefix patterns and check for closure of these patterns, while the backward directional extension is used to both check for closure and prune the search space.

Closed PROjected Window Lists [Huang et al. 2005] is an extension to the PROWL algorithm [Huang et al. 2004] that mines the set of closed frequent continuities. A *closed frequent continuity* is a continuity which has no proper supercontinuity with the same support. This is defined as follows: given two continuities $P = [p_1, p_2, \ldots, p_u]$ and $P' = [p'_1, p'_2, \ldots, p'_v]$ then P is a supercontinuity of P' (and therefore P' is a subcontinuity of P) if and only if, for each non-* pattern p'_j $(1 \leq j \leq w)$, $p'_j \subseteq p_{j+o}$ is true for some integer o and the integer o is termed the offset of P. This added constraint marks the only difference in the mining compared to that of PROWL.

In many situations, sequences are not presented to the user totally ordered and thus the search for closed partially ordered sequences can be important [Casas-Garriga 2005; Pei et al. 2005, 2006]. Frecpo (or Frequent closed partial *o*rders) [Pei et al. 2006] mines strings to produce representative sequences where the order of some elements may not be fixed and thus can provide a more complete and in some cases a smaller set of useful results.

6.2. Hybrid Methods

DISC - DIrect Sequence Comparison [Chiu et al. 2004] combines the candidate sequence and pruning strategy, database partitioning and projection with a strategy that recognizes the frequent sequences of a specific length k without having to compute the support of the nonfrequent sequences. This is achieved by using k-sorted databases to find all frequent k-sequences, which skips many of the nonfrequent k-sequences by checking only the conditional k-minimum subsequences. The basic DISC strategy is as follows. The first position in a k-sorted database is termed the *candidate k-sequence* and denoted by α_1. Further, given a minimum support count of δ, the k-minimum subsequence at the δ-th position is denoted α_δ. These two candidates are compared and if they are equal then α_1 is frequent since the first δ customer sequences in the

k-sorted database take α_1 as their k-minimum subsequence and if $\alpha_1 < \alpha_\delta$ then α_1 is nonfrequent and all subsequences up to and including α_δ can be skipped. This process is then repeated for the next k-minimum subsequence in the resorted k-sorted database. The proposed algorithm uses a partitioning method similar to SPADE [Zaki 2001b], SPAM [Ayres et al. 2002], and PrefixSpan [Pei et al. 2001b] for generating frequent 2-sequences and 3-sequences and then employs the DISC strategy to generate the remaining frequent sequences.

6.3. Approximate Methods

ApproxMAP - Approximate Multiple Alignment Pattern mining [Kum et al. 2002; Kum et al. 2007b] mines consensus patterns from large databases by a two-step strategy. A consensus pattern is one shared by many sequences and covers many short patterns but may not be exactly contained in any one sequence. It has two steps.

(1) In the first step sequences are clustered by similarity and then from these clusters consensus patterns are mined directly through a process of multiple alignment. To enable the clustering of sequences a modified version of the *hierarchical edit distance* metric is used in a density-based clustering algorithm. Once this stage is completed, each cluster contains similar sequences and a summary of the general pattern of each cluster.
(2) The second step is trend analysis. First the density for each sequence is calculated and the sequences in each cluster are sorted in density descending order, second the multiple alignment process is carried out using a weighted sequence, which records the statistics of the alignment of the sequences in the cluster. Consensus patterns are then selected based on the parts of the weighted sequence that are shared by many sequences in the cluster using a *strength threshold* (similar to support) to remove items whose strength is less than the threshold.

Kum et al. [2007b] provide a comparison showing the reduction in the number of patterns produced under the approximate model compared to the traditional support model.

ProMFS (Probabilistic Mining Frequent Sequences) is based on estimated statistical characteristics of the appearance of elements of the sequence being mined and their order [Tumasonis and Dzemyda 2004]. The main thrust of this technique is to generate a smaller sequence based on these estimated characteristics and then make decisions about the original sequence based on analysis of the shorter one. Once the generation of the shorter sequence has concluded then analysis can be undertaken using GSP [Srikant and Agrawal 1996] or any other suitable algorithm.

Fuzzy logic has also been applied to the problem [Fiot et al. 2007; Hu et al. 2003, 2004; Hong et al. 2001]. For example, Fiot et al. [2007] propose three algorithms, based around the PSP algorithm [Masseglia et al. 1998] for discovering sequential patterns from numeric data that differ according to the scale of fuzziness desired by the user, while Hu et al. [2003] propose a fuzzy grids approach based around the Apriori class of miner.

6.4. Parallel Algorithms

With the discovery of sequential patterns becoming increasingly useful and the size of the available datasets growing rapidly there is a growing need for both efficient and scalable algorithms. To this end a number of algorithms have been developed to take advantage of distributed memory parallel computer systems and in doing so their computational power. For example, Demiriz and Zaki [2002] developed webSPADE, a modification of the SPADE algorithm [Zaki 2001b], to enable it to be run on shared-memory parallel computers and was developed to analyze Web log data that had been cleaned and stored in a data warehouse.

Guralnik and Karypis [2004] developed two static distribution algorithms, based on the tree projection algorithm for frequent itemset discovery [Agrawal et al. 1999], to take advantage of either data or task parallelism. The data-parallel algorithm partitions the database across different processors whereas the task-parallel algorithm partitions the lattice of frequent patterns, and both take advantage of a dynamic load-balancing scheme.

The Par-CSP (Parallel Closed Sequential Pattern mining) algorithm [Cong et al. 2005] is based on the BIDE algorithm [Wang and Han 2004] and mines closed sequential patterns on a distributed memory system. The process is divided into independent tasks to minimize inter-processor communication while using a dynamic scheduling scheme to reduce processor idle time. This algorithm also uses a load-balancing scheme.

6.5. Other Methods

Antunes and Oliveira [2003] use a modified string edit distance algorithm to detect whether a given sequence approximately matches a given constraint, expressed as a regular language, based on a *generation cost*. The algorithm, *ϵ-accepts*, can be used with any algorithm that uses regular expressions as a constraint mechanism, that is, the SPIRIT [Garofalakis et al. 1999] family of algorithms.

Yang et al. [2002] developed a method that uses a *compatibility matrix*, a conditional probability matrix that specifies the likelihood of symbol substitution, in conjunction with a *match* metric (*aggregated amount of occurrences*) to discover long sequential patterns using primarily gene sequence data as the input.

Hingston [2002] viewed the problem of sequence mining as one of inducing a finite state automaton model that is a compact summary of the sequential dataset in the form of a generative model or grammar and then using that model to answer queries about the data.

All of the algorithms discussed thus far are either 1- or 2-dimensional but some research has been conducted on multidimensional sequence mining. Yu and Chen [2005] introduce two algorithms, the first of which modifies the traditional Apriori algorithm [Agrawal and Srikant 1994] and the second by modifying the PrefixSpan algorithm [Pei et al. 2001b]. Pinto et al. [2001] propose a theme of multidimensional sequential pattern mining, which integrates both multidimensional analysis and sequential pattern mining and further explores feasible combinations of these two methods and makes recommendations on the proper selection with respect to particular datasets.

7. INCREMENTAL MINING ALGORITHMS

The ever-increasing amount of data being collected has also introduced the problem of how to handle the addition of new data and the possible nonrelevance of older data. In common with association rule mining, there have been many methods proposed to deal with this.

Incremental Discovery of Sequential Patterns was an early algorithm where the database can be considered in two modes [Wang and Tan 1996; Wang 1997].

(1) A single long sequence of events mode is one in which a frequent sequential pattern is a subsequence and an update is either the insertion or deletion of a subsequence at either end.
(2) A collection of sequences and a frequent sequential pattern as a subsequence and an update is the insertion or deletion of a complete sequence is the other mode.

Interactive Sequence Mining [Parthasarathy et al. 1999] is built on the SPADE algorithm [Zaki 2001b] and aims to minimize the I/O and computational requirements inherent in incremental updates and is concerned with both the addition of new customers and new transactions. It consists of two phases, the first the updating of the supports of elements in *Negative Border*[2] (*NB*) and *Frequent Sequences* (*FS*) and the second the addition to *NB* and *FS* beyond the first phase. The algorithm also maintains an *Incremental Sequence Lattice ISL*, which consists of the frequent sequences plus sequences in the *NB* in the original database, along with the supports for each.

Incremental Sequence Extraction [Masseglia et al. 2000] is an algorithm for updating frequent sequences where new transactions are appended to customers who already exist in the original database. This is achieved by using information from the frequent sequences of previous mining runs. Firstly, if k is the longest frequent sequence in *DB* (the original database), it finds all new frequent sequences of size $j \leq (k+1)$ considering three type of sequences:

(1) those that are embedded in DB that become frequent due to an increase in support due to the increment database *db*,
(2) frequent sequences embedded in *db*,
and
(3) those sequences of *DB* that become frequent by the addition of an item from a transaction in *db*,

and second, it finds all frequent sequences of size $j > (k + 1)$. In the first pass the algorithm finds frequent 1-sequences by considering all items in *db* that occur once and then by considering all items in *DB* to determine which items of *db* are frequent in U $(DB \cup db)$, which are denoted L_1^{db}. These frequent 1-sequences can be used as seeds to discover new frequent sequences of the following types:

(1) previous frequent sequences in *DB* which can be extended by items of L_1^{db};
(2) frequent subsequences in *DB* which are predecessor items in L_1^{db};
(3) candidate sequences generated from L_1^{db}.

This process is continued iteratively (since after the first step frequent 2-sequences are obtained to be used in the same manner as described before) until no more candidates are generated.

Incrementally/Decreasingly Updating Sequences are two algorithms for discovering frequent sequences when new data is added to an original database and for deleting old sequences that are no longer frequent after the addition of new data respectively [Zheng et al. 2002]. The IUS algorithm makes use of the negative border sequences and the frequent sequences of the original database in a similar fashion to the ISM algorithm [Parthasarathy et al. 1999] but introduces an added support threshold for the negative border. This added support metric (*Min_nbd_supp*) means that only those sequences whose support is between the original *min_supp* value and *Min_nbd_supp* can be members of the *negative border* set. Both prefixes and suffixes of the original frequent sequences are extended to generate candidates for the updated database. DUS recomputes the *negative border* and frequent sequences in the updated database based on the results of a previous mining run and prunes accordingly.

Zhang et al. [2002] created two algorithms based on the nonincremental versions from which they are named: GSP [Srikant and Agrawal 1996] and MFS [Zhang et al. 2001]. The problem of incremental maintenance here is one of updates via insertions of new sequences and deletions via removal of old sequences and follows a similar pattern to previous algorithms by taking advantage of the information from a previous mining

[2]The *negative border* is the collection of all sequences that are not frequent but both of whose generating subsequences are frequent.

run. The modifications to both GSP and MFS are based on three observations [Zhang et al. 2002].

(1) The portion of the database that has not been changed, which in many cases is the majority, need not be processed since the support of a frequent sequence in the updated database can be deduced by only processing the inserted and deleted customer sequences.
(2) If a sequence is infrequent in the old database then in order for it to become frequent in the updated database, its support in the inserted portion has to be large enough and therefore it is possible to determine whether a candidate should be considered by only using the portion of the database that has changed.
(3) If both the old database and the new updated database share a nontrivial set of common sequences, then the set of frequent sequences that have already been mined will give a good indication of likely frequent sequences in the updated database.

These observations are used to develop pruning tests in order to minimize the scanning of the old database to count the support of candidate sequences.

Incremental Sequential Pattern mining [Cheng et al. 2004] is based on the nonincremental sequential pattern mining algorithm PrefixSpan [Pei et al. 2001b] and uses a similar approach to Zheng et al. [2002] by buffering *semifrequent sequences* (*SFS*) for use as candidates for newly appearing sequences in the updated database. This buffering is achieved by using a *buffer ratio* $\mu \leq 1$ to lower the *min_supp* and then maintaining the set of sequences, in the original database, that meet this modified *min_supp*. The assumption is that the majority of the frequent subsequences introduced by the updated part of the database (D') would come from these *SFS* or are already frequent in the original database (D) and therefore according to Cheng et al. [2004] the SFS' and FS' in D' are derived from the following cases:

(1) A pattern which is frequent in D is still frequent in D'.
(2) A pattern which is semifrequent in D becomes frequent in D'.
(3) A pattern which is semifrequent in D is still semifrequent in D'.
(4) An appended database Δdb brings new items (either frequent or semifrequent).
(5) A pattern which is infrequent in D becomes frequent in D'.
(6) A pattern which is infrequent in D becomes semifrequent in D'.

For cases (1)–(3) the information required to update the support and project D' to find all frequent/semifrequent sequences is already in FS and SFS and is therefore a trivial exercise.
Case (4): *Property*: An item which does not appear in D and is brought by Δdb has no information in *FS* or *SFS*.
Solution: Scan the database *LDB*[3] for single items and then use any new frequent item as a prefix to construct a projected database and recursively apply the PrefixSpan method to discover frequent and semifrequent sequences.
Case (5): *Property*: If an infrequent sequence p' in D becomes frequent in D', all of its prefix subsequences must also be frequent in D'. Then at least one of its prefix subsequences p is in *FS*.
Solution: Start from its frequent prefix p in FS and construct a p-projected database on D' and use the PrefixSpan method to discover p'.
Case (6): *Property*: If an infrequent sequence p' becomes semifrequent in D', all of its prefix subsequences must be either frequent of semifrequent. Then at least one of its prefix subsequences, p, is in FS or SFS.

[3]The *LDB* is the set of sequences in D' which are appended with items/itemsets

Solution: Start from its prefix p in FS or SFS and construct a p-projected database on D' and use the PrefixSpan method to discover p'.

The task of the algorithm is thus, given an original database D, and appended database D', a threshold *min_supp*, a buffer ratio μ, a set of frequent sequences FS, and a set of semifrequent sequences, SFS, to find all FS' in D' [Cheng et al. 2004]. This is a two-step process in which LDB is scanned for single items (Case (4)) and then every pattern in FS and SFS in LDB are checked to adjust the support of them. If a pattern becomes frequent add it to FS'. If it meets the projection condition ($supp_{LDB}(p) \geq (1 - \mu) * min\_supp$) then use it as a prefix to project a database as in Case (5) and if the pattern is semifrequent add it to SFS'. Optimization techniques dealing with pattern matching and projected database generation are also applied.

Nguyen et al. [2005] found that IncSpan may fail to mine the *complete* set of sequential patterns from an updated database. Furthermore, they clarify the weaknesses in the algorithm and provide a solution, IncSpan+ which rectifies them. The weaknesses identified are for Cases (4)–(6) and proofs are given, in the form of counter examples. For Case (4) they prove that not all single frequent/semifrequent items can be found by scanning LDB and propose a solution that scans the entire D' for new single items. For Case (5) they show that it is possible that none of the prefix subsequences p of an infrequent sequence, p' in D that becomes frequent in D' is in FS and propose to not only mine the set of FS but also those of SFS. For Case (6) it is shown that it is possible that none of the prefix subsequences of p of an infrequent sequence, p' in D that becomes semifrequent in D' is in FS or SFS and propose to not only mine the set of FS but also those of SFS based on the projection condition. By implementing these changes IncSpan+ not only guarantees the correctness of the incremental mining result, but also maintains the complete set of semifrequent sequences for future updates [Nguyen et al. 2005].

8. AREAS OF RELATED RESEARCH

8.1. Streaming Data

Mining data streams is an area of related research that is still in its infancy, however, there is a significant and growing interest in the creation of algorithms to accommodate different datasets. Among these researchers are Lin et al. [2003] who look at a symbolic representation of time series and the implications with respect to streaming algorithms. Giannella et al. [2003] who mine frequent patterns in data stream using multiple time granularities, and Cai et al. [2004] and Teng et al. [2003] who use multidimensional regression methods to solve the problem. Laur et al. [2005] evaluate a statistical technique that biases the estimation of the support of sequential patterns, so as to maximize either the precision or the recall and limits the degradation of the any other criteria and Marascu and Masseglia [2005] use the sequence alignment method ApproxMAP [Kum et al. 2002] to find patterns in Web usage data streams. A review of the field of mining data streams can be found elsewhere [Gaber et al. 2005; Aggarwal 2007].

8.2. String Matching and Searching

An older yet related field is that of approximate string matching and searching, where the strings are viewed as sequences of tokens. Research in this area has been active for some time and includes not only algorithms for string matching using regular expressions [Hall and Dowling 1980; Aho 1990; Breslauer and Gąsieniec 1995; Bentley and Sedgewick 1997; Landau et al. 1998; Sankoff and Kruskal 1999; Chan et al. 2002; Amir et al. 2000], but also algorithms in the related area of edit distance [Wagner and Fischer 1974; Tichy 1984; Bunke and Csirik 1992; Oommen and Loke 1995; Oommen and Zhang 1996; Cole and Hariharan 1998; Arslan and Egecioglu 1999, 2000; Cormode and Muthukrishnan 2002; Batu et al. 2003; Hyyrö 2003. For a survey on this field refer to the work of Navarro [2001].

There is, however, no reason why text cannot be viewed in the same way and Ahonen et al. [1997, 1998] and Rajman and Besançon [1998] have applied similar techniques, based on generalized episodes and episode rules, to text analysis tasks.

8.3. Rule Inference

The purpose of generating frequent sequences, no matter which techniques are used, is to be able to infer some type of *rule*, or *knowledge*, from the results. For example, given the sequence $A \ldots B \ldots C$ occurring in a string multiple times, rules such as: token (or event) A appears (occurs) then B and then C, can be expressed. It has been argued by Padmanabhan and Tuzhilin [1996] that these types of inference, and hence the temporal patterns, have a limited expressive power. For this reason mining for sequences and the generation of rules based on first-order temporal logic (FOTL), carried out by Padmanabhan and Tuzhilin [1996], has extended the work of Mannila et al. [1995] to include inferences of the type *Since, Until, Next, Always, Sometimes, Before, After, and While*, by searching the database for specific patterns that satisfy a particular Temporal Logic Formula (TLF). One disadvantage to this approach is that no intermediate results are obtained and thus any mining for a different pattern must be conducted on the complete database, which could incur a significant overhead.

Höppner [2001] and Höppner and Klawonn [2002] use modified rule semantics, J-Measure and rule specialization to find temporal rules from a set of frequent patterns in a state sequence. The method described uses a windowing approach, similar to that used to discover frequent episodes, and then imposes Allen's interval logic [Allen 1983] to describe rules that exist within these temporal patterns. Kam and Fu [2000] deal with temporal data for events that last over a period of time and introduce the concept of temporal representation and discuss the view that this can be used to express relationships between interval-based events, using Allen's temporal logic. Sun et al. [2003] use "Event-oriented patterns" in order to generate rules, with a given minimum confidence, that are in the form of $LHS \xrightarrow{T} e,\ conf(\theta)$.

9. DISCUSSION AND FUTURE DIRECTIONS

This article has surveyed the field of sequential pattern mining with a focus on the algorithmic approaches required to meet different requirements and the inclusion of constraints, and counting methods for windowing-based algorithms.

A number of future areas of research are likely to be explored in the future.

(1) While the rules produced from the majority of approaches are simple, in the sense they do not take into account the use of temporal logic algebras and their derivatives, some algorithms are starting to produce rules that are more expressive [Padmanabhan and Tuzhilin 1996; Sun et al. 2003]. This is an area that will likely develop further in the future.
(2) Some sequences are an example of relative ordering in time. However, with few exceptions, pinning some of the events to absolute time points and the implication this has for pattern mining has not been investigated. For example, there are few algorithms that could state that a given sequence occurs on Mondays but not on other days.
(3) More generally, accommodating interval semantics as well as point-based tokens in the event stream would provide richer rulesets. This might simply be done by adding the start and end of intervals in as tokens in the sequence but more powerful and efficient algorithms might also be developed.

(4) The GSP algorithm [Srikant and Agrawal 1996] discussed the handling of events that formed a part of a hierarchy. However, this is an aspect that has not been pursued. Indeed, the confluence of ontologies/taxonomies and data mining is an area that has to date only received minor attention.
(5) Finally, with the improvements in incremental sequential pattern mining and the growing field of stream data mining the need to express rules with a more expressive nature is becoming more appealing and has the potential to avail the sequence mining paradigm to more diverse and complex domains.

REFERENCES

AGGARWAL, C. C. 2007. *Data Streams: Models and Algorithms.* Springer, New York.

AGRAWAL, R. AND SRIKANT, R. 1994. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*. J. B. Bocca, M. Jarke, and C. Zaniolo, Eds., Morgan Kaufmann, 487–499.

AGRAWAL, R. AND SRIKANT, R. 1995. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE'95).* P. S. Yu and A. S. P. Chen, Eds., IEEE Computer Society Press, 3–14.

AGRAWAL, R. C., AGGARWAL, C. C., AND PRASAD, V. V. V. 1999. A tree projection algorithm for generation of frequent itemsets. In *High Performance Data Mining Workshop.* ACM Press.

AHO, A.1990. *Algorithms for Finding Patterns in Strings.* Vol. A: Algorithms and Complexity. MIT Press, Cambridge, MA, 255–300.

AHONEN, H., HEINONEN, O., KLEMETTINEN, M., AND VERKAMO, A. I. 1997. Applying data mining techniques in text analysis. Tech. rep. C-1997-23, Department of Computer Science, University of Helsinki.

AHONEN, H., HEINONEN, O., KLEMETTINEN, M., AND VERKAMO, A. I. 1998. Applying data mining techniques for descriptive phrase extraction in digital document collections. In *Proceedings of the Advances in Digital Libraries Conference.* IEEE Computer Society, 2.

ALBERT-LORINCZ, H. AND BOULICAUT, J.-F. 2003a. A framework for frequent sequence mining under generalized regular expression constraints. In *Proceedings of the 2nd International Workshop on Inductive Databases*. KDID, J.-F. Boulicaut and S. Dzeroski, Eds., 2–16.

ALBERT-LORINCZ, H. AND BOULICAUT, J.-F. 2003b. Mining frequent sequential patterns under regular expressions: A highly adaptive strategy for pushing contraints. In *Proceedings of the 3rd SIAM International Conference on Data Mining.* D. Barbar'a and C. Kamath, Eds., SIAM.

ALLEN, J. F. 1983. Maintaining knowledge about temporal intervals. *Comm. ACM 26,* 11,832–843.

AMIR, A., LEWENSTEIN, M., AND PORAT, E. 2000. Faster algorithms for string matching with k mismatches. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 794–803.

ANTUNES, C. AND OLIVEIRA, A. L. 2004. Sequential pattern mining with approximated constraints. In *Proceedings of the International Conference on Applied Computing.*

ARSLAN, A. N. AND EGECIOGLU, O. 1999. An efficient uniform-cost normalized edit distance algorithm. In *Proceedings of the 6th Symposium on String Processing and Information Retrieval (SPIRE'99).* IEEE Computer Society, 8–15.

ARSLAN, A. N. AND EGECIOGLU, O. 2000. Efficient algorithms for normalized edit distance. *J. Discr. Algor. 1,* 1, 3–20.

AYRES, J., FLANNICK, J., GEHRKE, J., AND YIU, T. 2002. Sequential pattern mining using a bitmap representation. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM Press, 429–435.

BATU, T., ERGÜN, F., KILIAN, J., MAGEN, A., RASKHODNIKOVA, S., RUBINFELD, R., AND SAMI, R. 2003. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of the 35th ACM Symposium on Theory of Computing.* ACM Press, 316–324.

BAYARDO, R. J. AND AGRAWAL, R. 1999. Mining the most interesting rules. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining.* S. Chaudhuri and D. Madigan, Eds., ACM Press, 145–154.

BENTLEY, J. L. AND SEDGEWICK, R. 1997. Fast algorithms for sorting and searching strings. In *Proceedings of the 8th Annual ACM/SIAM Symposium on Discrete Algorithms.* SIAM, 360–369.

BRESLAUER, D. AND GĄSIENIEC, L. 1995. Efficient string matching on coded texts. In *Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching.* Z. Galil and E. Ukkonen, Eds., Springer, 27–40.

BUNKE, H. AND CSIRIK, J. 1992. Edit distance of run-length coded strings. In *Proceedings of the ACM/SIGAPP Symposium on Applied Computing.* ACM Press, 137–143.

CAI, Y. D., CLUTTER, D., PAPE, G., HAN, J., WELGE, M., AND AUVIL, L. 2004. Maids: Mining alarming incidents from data streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* ACM Press, 919–920.

CASAS-GARRIGA, G. 2005. Summarizing sequential data with closed partial orders. In *Proceedings of the 5th SIAM International Conference on Data Mining.* H. Kargupta, J. Srivastava, and A. Chandrika Kamath, Eds., Vol. 119, 380–391.

CEGLAR, A. AND RODDICK, J. F. 2006. Association mining. *ACM Comput. Surv. 38*, 2.

CEGLAR, A., RODDICK, J. F., AND CALDER, P. 2003. *Guiding Knowledge Discovery Through Interactive Data Mining.* Idea Group Publishers, Hershey, PA, 45–87.

CHAKRABARTI, S., SARAWAGI, S., AND DOM, B. 1998. Mining surprising patterns using temporal description length. In *Proceedings of the 24th International Conference on Very Large Data Bases, (VLDB'98).* A. Gupta, O. Shmueli, and J. Widom, Eds. Morgan Kaufmann, 606–617.

CHAN, S., KAO, B., YIP, C. L., AND TANG, M. 2002. Mining emerging substrings. Tech. rep. TR-2002-11, HKU CSIS.

CHENG, H., YAN, X., AND HAN, J. 2004. Incspan: Incremental mining of sequential patterns in large database. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '04).* ACM Press, 527–532.

CHIU, D.-Y., WU, Y.-H., AND CHEN, A. L. P. 2004. An efficient algorithm for mining frequent sequences by a new strategy without support counting. In *Proceedings of the 20th International Conference on Data Engineering (ICDE'04).* IEEE Computer Society, 375–386.

COLE, R. AND HARIHARAN, R. 1998. Approximate string matching: A simpler faster algorithm. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms.* SIAM, 463–472.

CONG, S., HAN, J., AND PADUA, D. A. 2005. Parallel mining of closed sequential patterns. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* R. Grossman, R. Bayardo, and K. P. Bennett, Eds., ACM, 562–567.

CORMODE, G. AND MUTHUKRISHNAN, S. 2002. The string edit distance matching problem with moves. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms.* SIAM, 667–676.

DEMIRIZ, A. AND ZAKI, M. J. 2002. webSPADE: A parallel sequence mining algorithm to analyze the web log data. In *Proceedings of the 2nd IEEE International Conference on Data Mining.*

EL-SAYED, M., RUIZ, C., AND RUNDENSTEINER, E. A. 2004. FS-miner: Efficient and incremental mining of frequent sequence patterns in web logs. In *Proceedings of the 6th ACM International Workshop on Web Information and Data Management (WIDM'04).* A. H. F. Laender, D. Lee, and M. Ronthaler, Eds., ACM, 128–135.

FIOT, C., LAURENT, A., AND TEISSEIRE, M. 2007. From crispness to fuzziness: Three algorithms for soft sequential pattern mining. *IEEE Trans. Fuzzy Syst. 15,* 6, 1263–1277.

FU, Y. AND HAN, J. 1995. Meta-Rule-Guided mining of association rules in relational databases. In *Proceedings of the 1st International Workshop on Integration of Knowledge Discovery with Deductive and Object-Oriented Databases (KDOOD'95).* 39–46.

GABER, M. M., ZASLAVSKY, A., AND KRISHNASWAMY, S. 2005. Mining data streams: A review. *SIGMOD Rec. 34,* 2, 18–26.

GAROFALAKIS, M. N., RASTOGI, R., AND SHIM, K. 1999. SPIRIT: Sequential pattern mining with regular expression constraints. In *Proceedings of the 25th International Conference on Very Large Databases (VLDB'99).* 223–234.

GIANNELLA, C., HAN, J., PEI, J., YAN, X., AND YU, P. S. 2003. Mining frequent patterns in data streams at multiple time granularities. In *Next Generation Data Mining*. H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha, Eds., 191–212.

GURALNIK, V. AND KARYPIS, G. 2004. Parallel tree-projection-based sequence mining algorithms. *Parallel Comput. 30,* 4, 443–472.

GURALNIK, V., WIJESEKERA, D., AND SRIVASTAVA, J. 1998. Pattern directed mining of sequence data. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD '98).* R. Agrawal, P. E. Stolorz, and G. Piatetsky-Shapiro, Eds., AAAI Press, 51–57.

HALL, P. A. V. AND DOWLING, G. R. 1980. Approximate string matching. *ACM Comput. Surv. 12,* 4, 381–402.

HAN, J., CHENG, H., XIN, D., AND YAN, X. 2007. Frequent pattern mining: Current status and future directions. *Data Mining Knowl. Discov. 15,* 1, 55–86.

HAN, J., KOPERSKI, K., AND STEFANOVIC, N. 1997. GeoMiner: A system prototype for spatial data mining. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data (SIGMOD '97).* J. Peckham, Ed., ACM Press, 553–556.

Han, J. and Pei, J. 2000. Mining frequent patterns by pattern growth: Methodology and implications. *SIGKDD Explor. Newslett. 2,* 2, 14–20.

Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., and Hsu, M.-C. 2000a. Freespan: Frequent pattern-projected sequential pattern mining. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM Press, 355–359.

Han, J., Pei, J., and Yin, Y. 2000b. Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '00).* ACM, 1–12.

Hingston, P. 2002. Using finite state automata for sequence mining. In *Proceedings of the 25th Australasian Conference on Computer Science (ACSC'02).* Australian Computer Society, Inc., 105–110.

Hong, T. P., Lin, K. Y., and Wang, S. L. 2001. Mining fuzzy sequential patterns from multiple-item transactions. In *Proceedings of the Joint 9th IFSA World Congress and 20th NAFIPS International Conference.* Vol. 3., IEEE, 1317–1321.

Hoppner, F. 2001. Discovery of temporal patterns. Learning rules about the qualitative behaviour of time series. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'01).* 192-203.

Hoppner, F. and Klawonn, F. 2002. Finding informative rules in interval sequences. *Intell. Data Anal. 6,* 6, 237–255.

Hsu, C. M., Chen, C. Y., Liu, B. J., Huang, C. C., Laio, M. H., Lin, C. C., and Wu, T. L. 2007. Identification of hot regions in protein-protein interactions by sequential pattern mining. *BMC Bioinf. 8,* 5, 8.

Hu, Y. C., Chen, R. S., Tzeng, G. H., and Shieh, J. H. 2003. A fuzzy data mining algorithm for finding sequential patterns. *Int. J. Uncert., Fuzziness Knowl. Based Syst. 11,* 2, 173–194.

Hu, Y. C., Tzeng, G. H., and Chen, C. M. 2004. Deriving two-stage learning sequences from knowledge in fuzzy sequential pattern mining. *Inf. Sci. 159,* 1-2, 69–86.

Huang, K.-Y., Chang, C.-H., and Lin, K.-Z. 2004. PROWL: An efficient frequent continuity mining algorithm on event sequences. In *Proceedings of the 6th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'04).* Y. Kambayashi and W. Wöß, Eds., Lecture Notes in Computer Science, vol. 3181, Springer, 351–360.

Huang, K.-Y., Chang, C.-H., and Lin, K.-Z. 2005. ClosedPROWL: Efficient mining of closed frequent continuities by projected window list technology. In *Proceedigns of the SIAM International Conference on Data Mining.*

Hyyro, H. 2003. A bit-vector algorithm for computing levenshtein and damerau edit distances. *Nordic J. Comput. 10,* 1, 29–39.

Joshi, M. V., Karypis, G., and Kumar, V. 1999. Universal formulation of sequential patterns. Tech. rep. 99-21, Department of Computer Science, University of Minnesota.

Kam, P.-S. and Fu, A. W.-C. 2000. Discovering temporal patterns for interval-based events. In *Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery (DaWaK '00).* Y. Kambayashi, M. K. Mohania, and A. M. Tjoa, Eds., Lecture Notes in Computer Science, vol. 1874., Springer, 317–326.

Kum, H.-C., Chang, J. H., and Wang, W. 2007a. Benchmarking the effectiveness of sequential pattern mining methods. *Data Knowl. Engin. 60,* 1, 30–50.

Kum, H.-C., Chang, J. H., and Wang, W. 2007B. Intelligent sequential mining via alignment: Optimization techniques for very large databases. In *Proceedings of the 11th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining. (PAKDD'07).* Springer, 587 – 597.

Kum, H.-C., Pei, J., Wang, W., and Duncan, D. 2002. ApproxMAP: Approximate mining of consensus sequential patterns. In *Mining Sequential Patterns from Large Data Sets,* W. Wang and J. Yang, Eds. Vol. 28., Springer.

Landau, G. M., Myers, E. W., and Schmidt, J. P. 1998. Incremental string comparison. *SIAM J. Comput. 27,* 2, 557–582.

Laur, P.-A., Symphor, J.-E., Nock, R., and Poncelet, P. 2005. Mining sequential patterns on data streams: A near-optimal statistical approach. In *Proceedigns of the 2nd International Workshop on Knowledge Discovery from Data Streams.*

Lin, J., Keogh, E., Lonardi, S., and Chiu, B. 2003. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery.* ACM Press, 2–11.

Luo, C. and Chung, S. M. 2004. A scalable algorithm for mining maximal frequent sequences using sampling. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '04).* IEEE Computer Society, 156–165.

Mabroukeh, N. R. and Ezeife, C. I. 2010. A taxonomy of sequential pattern mining algorithms. *ACM Comput. Surv. 43,* 1, 3.

Mannila, H. and Toivonen, H. 1996. Discovering generalized episodes using minimal occurrences. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD '96)*. AAAI Press, 146–151.

Mannila, H., Toivonen, H., and Verkamo, A. I. 1995. Discovering frequent episodes in sequences. In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining (KDD '95)*. U. M. Fayyad and R. Uthurusamy, Eds., AAAI Press, 210–215.

Mannila, H., Toivonen, H., and Verkamo, A. I. 1997. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov. 1*, 3, 259–289.

Marascu, A. and Masseglia, F. 2005. Mining sequential patterns from temporal streaming data. In *Proceedings of the 1stECML/PKDD Workshop on Mining Spatio-Temporal Data (MSTD'05),* held in conjunction with the *9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'05).*

Masseglia, F., Cathala, F., and Poncelet, P. 1998. The PSP approach for mining sequential patterns. In *Proceedings of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'98).* Lecture Notes in Artificial Intelligence, vol. 1510., Springer, 176–184.

Masseglia, F., Poncelet, P., and Teisseire, M. 2000. Incremental mining of sequential patterns in large databases. Tech. rep., LIRMM.

Mooney, C. H. and Roddick, J. F. 2006. Marking time in sequence mining. In *Proceedings of the Australasian Conference on Data Mining and Analystics (AusDM '06).* P. Christen, P. Kennedy, J. Li, S. Simoff, and G. Williams, Eds., Vol. 61.

Navarro, G. 2001. A guided tour to approximate string matching. *ACM Comput. Surv. 33,* 1, 31–88.

Ng, R. T., Lakshmanan, L. V. S., Han, J., and Pang, A. 1998. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'98).* ACM, 13–24.

Nguyen, S. N., Sun, X., and Orlowska, M. E. 2005. Improvements of incspan: Incremental mining of sequential patterns in large database. In *Proceedings of the 9th Pacific-Asia Conference (PAKDD'05).* T. B. Ho, D. Cheung, and H. Liu, Eds., Vol. 3518., Springer, 442–451.

Oommen, B. J. and Loke, R. K. S. 1995. Pattern recognition of strings with substitutions, insertions, deletions and generalized transpositions. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics.* Vol. 2. 1154–1159.

Oommen, B. J. and Zhang, K. 1996. The Normalized String Editing problem revisited. *IEEE Trans. Pattern Anal. Mach. Intell. 18,* 6, 669–672.

Orlando, S., Perego, R., and Silvestri, C. 2004. A New Algorithm for gap constrained sequence mining. In *Proceedings of the ACM Symposium on Applied Computing (SAC).* ACM Press, 540–547.

Ouh, J. Z., Wu, P. H., and Chen, M. S. 2001. Experimental results on a constrained based sequential pattern mining for telecommunication alarm data. In *Proceedings of the 2nd International Conference on Web Information Systems Engineering (WISE'01).* IEEE Computer Society, 186–193.

Padmanabhan, B. and Tuzhilin, A. 1996. Pattern discovery in temporal databases: A temporal logic approach. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining.* E. Simoudis, J. Han, and U. Fayyad, Eds., AAAI Press, 351–354.

Pan, F., Cong, G., Tung, A. K. H., Yang, J., and Zaki, M. J. 2003. Carpenter: finding closed patterns in long biological datasets. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03).* ACM Press, 637–642.

Parthasarathy, S., Zaki, M. J., Ogihara, M., and Dwarkadas, S. 1999. Incremental and interactive sequence mining. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM).* ACM, 251–258.

Pei, J., Han, J., and Lakshmanan, L. V. S. 2001a. Mining frequent itemsets with convertible constraints. In *Proceedings of the 17th International Conference on Data Engineering.* IEEE Computer Society, 433–442.

Pei, J., Han, J., and Mao, R. 2000a. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proceedings of the ACM SIGMOD International Workshop on Data Mining.* ACM Press, 21–30.

Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., and Hsu, M.-C. 2001b. PrefixSpan mining sequential patterns efficiently by prefix projected pattern growth. In *Proceedings of the International Conference of Data Engineering (ICDE'01).* 215–226.

Pei, J., Han, J., Mortazavi-Asl, B., and Zhu, H. 2000b. Mining access patterns efficiently from web logs. In *Proceedings of the 4th Pacific-Asia Conference (PAKDD'00).* Lecture Notes in Computer Science, vol. 1805., Springer, 396–407.

PEI, J., HAN, J., AND WANG, W. 2002. Mining sequential patterns with constraints in large databases. In *Proceedings of the 11th International Conference on Information and Knowledge Management.* ACM Press, 18–25.

PEI, J., HAN, J., AND WANG, W. 2007. Constraint-Based sequential pattern mining: The pattern-growth methods. *J. Intell. Inf. Syst. 28,* 2, 133–160.

PEI, J., LIU, J., WANG, H., WANG, K., YU, P. S., AND WANG, J. 2005. Efficiently mining frequent closed partial orders. In *Proceedings of the 5th IEEE International Conference on Data Mining*. IEEE, 753–756.

PEI, J., WANG, H., LIU, J., WANG, K., WANG, J., AND YU, P. S. 2006. Discovering frequent closed partial orders from strings. *IEEE Trans. Knowl. Data Engin. 18,* 11, 1467–1481.

PINTO, H., HAN, J., PEI, J., WANG, K., CHEN, Q., AND DAYAL, U. 2001. Multi-Dimensional sequential pattern mining. In *Proceedings of the 10th International Conference on Information and Knowledge Management.* ACM Press, 81–88.

RAJMAN, M. AND BESANÇ, ON, R. 1998. Text mining – knowledge extraction from unstructured textual data. In *Proceedings of the 6th Conference of International Federation of Classification Societies (IFCS'98).*

SANKOFF, D. AND KRUSKAL, J. B. 1999. *Time Warps, String Edits, and Macromolecules / The Theory and Practice of Sequence Comparison*. David Hume Series., Center for the Study of Language and Information, Stanford, CA.

SAVARY, L. AND ZEITOUNI, K. 2005. Indexed bit map (ibm) for mining frequent sequences. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'05).* A. Jorge, L. Torgo, P. Brazdil, R. Camacho, and J. A. Gama, Eds., Lecture Notes in Computer Science, vol. 3721., Springer, 659–666.

SENO, M. AND KARYPIS, G. 2001. LPMiner: An algorithm for finding frequent itemsets using length-decreasing support constraint. In *Proceedigns of the 1st IEEE Conference on Data Mining.*

SENO, M. AND KARYPIS, G. 2002. SLPMiner: An algorithm for finding frequent sequential patterns using length-decreasing support. Tech. rep. 02-023, University of Minnesota.

SENO, M. AND KARYPIS, G. 2005. Finding frequent patterns using length-decreasing support constraints. *IEEE Trans. Knowl. Data Engin. 10,* 3, 197–228.

SRIKANT, R. AND AGRAWAL, R. 1996. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology (EDBT'96)*, P. M. G. Apers, M. Bouzeghoub, and G. Gardarin, Eds., Lecture Notes in Computer Science, vol. 1057. Springer, 3–17.

SRIVASTAVA, J., COOLEY, R., DESHPANDE, M., AND TAN, P.-N. 2000. Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explor. 1,* 2, 12–23.

SUN, X., ORLOWSKA, M. E., AND ZHOU, X. 2003. Finding event-oriented patterns in long temporal sequences. In *Proceedings of the 7th Pacific-Asia Conference (PAKDD'03),* K.-Y. Whang, J. J. and, K. S. and, and J. Srivastava, Eds., Lecture Notes in Computer Science, vol. 2637., Springer, 15–26.

TENG, W.-G., CHEN, M.-S., AND YU, P. S. 2003. A regression-based temporal pattern mining scheme for data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLBD '03),* J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, and A. Heuer, Eds., Morgan Kaufmann, 93–104.

TICHY, W. F. 1984. The string-to-string correction problem with block moves. *ACM Trans. Comput. Syst. 2,* 4, 309–321.

TOIVONEN, H. 1996. Discovery of frequent patterns in large data collections. Tech. rep. a-1996-5, Department of Computer Science, University of Helsinki.

TUMASONIS, R. AND DZEMYDA, G. 2004. The probabilistic algorithm for mining frequent sequences. In *Proceedings of the Conference on Advances in Databases and Information Systems (ADBIS).*

WAGNER, R. A. AND FISCHER, M. J. 1974. The string-to-string correction problem. *J. ACM 21,* 1, 168–173.

WANG, J. AND HAN, J. 2004. Bide: Efficient mining of frequent closed sequences. In *Proceedings of the International Conference on Data Engineering (ICDE'04).*

WANG, J., HAN, J., AND PEI, J. 2003. CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* L. Getoor, T. E. Senator, P. Domingos, and C. Faloutsos, Eds., ACM Press, 236–245.

WANG, K. 1997. Discovering patterns from large and dynamic sequential data. *J. Intell. Inf. Syst. 9,* 1, 33–56.

WANG, K. AND TAN, J. 1996. Incremental discovery of sequential patterns. In *Proceedings of the ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery.*

WANG, K., XU, Y., AND YU, J. X. 2004. Scalable sequential pattern mining for biological sequences. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management (CIKM'04).* ACM, 178–187.

WU, P. H., PENG, W. C., AND CHEN, M. S. 2001. Mining sequential alarm patterns in a telecommunication database. In *Databases in Telecommunications II*, W. Jonker, Ed., Lecture Notes in Computer Science, vol. 2209., Springer, 37–51.

YAN, X., HAN, J., AND AFSHAR, R. 2003. CloSpan: Mining closed sequential patterns in large datasets. In *Proceedings of the International Conference on Data Mining (SDM'03).*

YANG, J., WANG, W., YU, P. S., AND HAN, J. 2002. Mining long sequential patterns in a noisy environment. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'02).*

YANG, Z. AND KITSUREGAWA, M. 2005. LAPIN-SPAM: An improved algorithm for mining sequential pattern. In *Proceedings of the 21st International Conference on Data Engineering Workshops (ICDEW'05).* IEEE Computer Society, 1222.

YANG, Z., WANG, Y., AND KITSUREGAWA, M. 2005. LAPIN: Effective sequential pattern mining algorithms by last position induction. In *Proceedings of the 21st International Conference on Data Engineering (ICDE'05).*

YU, C.-C. AND CHEN, Y.-L. 2005. Mining sequential patterns from multidimensional sequence data. *IEEE Trans. Knowl. Data Engin. 17,* 1, 136–140.

ZAKI, M., LESH, N., AND OGIHARA, M. 1998. Planmine: Sequence mining for plan failures. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD'98)*, R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, Eds., ACM Press, 369–373.

ZAKI, M. J. 1998. Efficient enumeration of frequent sequences. In *Proceedings of the 7th International Conference on Information and Knowledge Management.* ACM Press, 68–75.

ZAKI, M. J. 2000. Sequence mining in categorical domains: Incorporating constraints. In *Proceedings of the 9th International Conference on Information and Knowledge Management (CIKM '00),* A. Agah, J. Callan, and E. Rundensteiner, Eds., ACM Press, 422–429.

ZAKI, M. J. 2001a. Parallel sequence mining on shared-memory machines. *J. Parallel Distrib. Comput. 61,* 3, 401–426.

ZAKI, M. J. 2001b. SPADE: An efficient algorithm for mining frequent sequences. *Mach. Learn. 42,* 1/2, 31–60.

ZAKI, M. J. AND HSIAO, C.-J. 2002. CHARM: An efficient algorithm for closed itemset mining. In *Proceedings of the 2nd SIAM International Conference on Data Mining (SDM'02).* R. L. Grossman, J. Han, V. Kumar, H. Mannila, and R. Motwani, Eds., SIAM, 457–473.

ZHANG, M., KAO, B., CHEUNG, D. W.-L., AND YIP, C. L. 2002. Efficient algorithms for incremental update of frequent sequences. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining.* 186–197.

ZHANG, M., KAO, B., YIP, C., AND CHEUNG, D. 2001. A GSP-based efficient algorithm for mining frequent sequences. In *Proceedings of the International Conference on Artificial Intelligence (ICAI'01).*

ZHAO, Q. AND BHOWMICK, S. S. 2003. Sequential pattern mining: A survey. Tech. rep., Nanyang Technological University, Singapore.

ZHENG, Q., XU, K., MA, S., AND LV, W. 2002. The algorithms of updating sequential patterns. In *Proceedings of the 5th International Workshop on High Performance Data Mining,* in conjunction with the *2nd SIAM Conference on Data Mining.*

Received September 2009; revised September 2011; accepted September 2011