Sequential Pattern Mining*

Tian-Rui Li[1], Yang Xu[1], Da Ruan[2], and Wu-ming Pan[3]

[1] School of Science, Southwest Jiaotong University, Chengdu 610031, P. R. China
{trli, xuyang}@swjtu.edu.cn
[2] Belgian Nuclear Research Centre (SCK•CEN), Boeretang 200, 2400 Mol, Belgium
druan@sckcen.be
[3] College of Software Engineering, Sichuan University, Chengdu 610065, P.R. China
fluedpan@hotmail.com

Summary. Sequential pattern discovery has emerged as an important research topic in knowledge discovery and data mining with broad applications. Previous research is mainly focused on investigating scalable algorithms for mining sequential patterns while less on its theoretical foundations. However, the latter is also important because it can help to use existing theories and methods to support more effective mining tasks. In this chapter, we conduct a systematic study on models and algorithms in sequential pattern analysis, especially discuss the existing algorithms' advantages and limitations. Then, we build the relation between the closed sequential patterns and fixed point, which can serve as a theoretical foundation of sequential patterns. Finally, we discuss its applications and outline the future research work.

1 Introduction

Data mining is mainly concerned with methodologies for extracting patterns from large data repositories. Sequential pattern mining, since its introduction in [1], has become an active research topic in data mining, with broad applications [2, 3]. For example, consider a Web access database at a popular site, where a Web user and Web page are regarded as an object and attribute respectively. The discovered patterns are the sequences of most frequently accessed pages at that site. This kind of information can help to improve a system design such as better a hyperlinked structure between the correlated pages and lead to better marketing decisions like strategic advertisement placement [2]. There are many other domains where sequence mining has been applied, which include discovering customer buying patterns in retail stores,

* This work was partially supported by the National Natural Science Foundation of China (NSFC) under the grant No.60474022.

Tian-Rui Li et al.: *Sequential Pattern Mining*, Studies in Computational Intelligence (SCI) **5**, 103–122 (2005)
www.springerlink.com

analysis of Web access databases, identifying plan failures, mining DNA sequences and gene structures, and finding network alarm patterns. Moreover, a deep understanding of efficient sequential pattern mining methods may also have strong implications on the development of efficient methods for mining frequent subtrees, lattices, subgraphs, and other structured patterns in large databases [4].

Many studies have been contributed to the efficient mining of sequential patterns in the literature, most of which was focused on developing efficient algorithms for finding all sequential patterns such as AprioriAll [1], GSP [5], SPADE [6], PrefixSpan [7] and so on. In addition, enormous sizes of available databases and possibly large number of mined sequential patterns demand efficient and scalable parallel algorithms. Therefore, several parallel algorithms such as HPSPM [8], pSPADE [9], TPF [10] were proposed and have good performance. Moreover, recent research on sequential pattern mining has progressed to closed sequential pattern mining, which can greatly reduce the number of frequent subsequences and improve the efficiency [4, 11].

Yet, the above work all assumes that the database is static, and a database updates requires rediscovering all the patterns by scanning the entire old and new database. Then, there is a need for efficient algorithms to update, maintain and manage the information discovered. Some incremental mining algorithms of sequential patterns were proposed, e.g. [12, 13, 14, 15, 16, 17], and perform significantly better than the naïve approach of mining the whole updated database from scratch.

However, a major common thread that runs through the vast majority of earlier work is the lack of user-controlled focus in the pattern mining process and then it demonstrates the need for novel pattern mining solutions. Recent feasible solutions are to incorporate user-specified constraints in sequential pattern mining, which enable the incorporation of user-controlled focus in the mining process and avoid overwhelming volume of potentially useless results [18, 19, 20].

Other work contributes on an extension of the problem of sequential pattern mining like mining cyclically repeated patterns [21], approximate mining of consensus sequential patterns [22], mining multidimensional sequential pattern [23], mining top-k closed sequential patterns [24], mining frequent Max sequential patterns [25], mining long sequential patterns in a noisy environment [26], mining hybrid sequential patterns and sequential rules [27], etc.

In this chapter, the models in sequential pattern mining are presented in Sect. 2. A systematic analysis on algorithms for mining sequential patterns is conducted in Sect. 3. In Sect. 4, a theoretical foundation of sequential pattern mining is provided. Its applications are discussed in Sect. 5. Our current study is summarized and some future research issues is also pointed out in Sect. 6.

2 Models of Sequential Patterns Mining

2.1 Original Model of Sequential Patterns Mining

The original model of mining sequential patterns was proposed in [1]. It can be stated as follows:

Let $I = \{i_1, i_2, \ldots, i_k\}$ be a set of all items. A subset of I is called an *itemset*. A *sequence* $s = \langle s_1, s_2, \ldots, s_m \rangle (s_i \subseteq I)$ is an ordered list. The *size*, $|s|$, of a sequence s is the number of itemsets in the sequence. The *length*, $l(s)$, is the total number of items in the sequence, namely, $l(s) = \sum_{i=1}^{m} |s_i|$. A sequence $\alpha = \langle \alpha_1, \alpha_2, \ldots, \alpha_m \rangle$ is a *sub-sequence* of another sequence $\beta = \langle \beta_1, \beta_2, \ldots, \beta_n \rangle$, denoted as $\alpha \sqsubseteq \beta$, (if $\alpha \neq \beta$, written as $\alpha \sqsubset \beta$), if and only if $\exists k_1, k_2, \ldots, k_m$, such that $1 \leq k_1 < k_2 < \ldots < k_m \leq n$ and $\alpha_1 \subseteq \beta_{k_1}, \alpha_2 \subseteq \beta_{k_2}, \ldots, \alpha_m \subseteq \beta_{k_m}$. We also call β is a super-sequence of α and β contains α. A *sequence database*, $D = \{d_1, d_2, \ldots, d_k\}$, is a set of sequences. Each sequence is associated with an *id*. For simplicity, say the *id* of d_i is i. $|D|$ represents the number of sequences in the database D. The *support* of a sequence α in a sequence database D is the number of sequences in D which contain α, $support(\alpha) = |\{d | d \in D \text{ and } \alpha \sqsubseteq d\}|$. Given a minimum support threshold, *min_sup*, the set of *frequent sequential patterns*, FS, includes all the sequences whose support is no less than *min_sup*. Given a sequence database and a user-specified *min_sup*, the problem of mining sequential patterns is to find all the frequent subsequences in the database.

The set of the closed frequent sequential pattern is defined as follows, $CS = \{\alpha \,|\alpha \in FS$ and $\nexists \beta \in FS$ such that $\alpha \sqsubseteq \beta$ and $support(\alpha) = support(\beta)\}$. Since CS includes no sequence which has a super-sequence with the same support, we have $CS \subseteq FS$. The problem of the closed sequence mining is to find CS above a minimum support threshold.

Example 1. Table 1 is a sample sequence database, referred as D when the context is clear. The alphabetic order is taken as the default lexicographical order. The set of *items* in the database is $\{a, b, c, d, e, f\}$. Obviously, $\langle (ce)(a)(af)(c) \rangle$ is a *sequence* in the database and its *size* is equal to 4. This whole sequence contributes only one to the *support* of (a) although item a appears more than once in it. The *support* of (a) is 5 in the database. If $min\_sup = 3$ (taken as default in this chapter), CS, FS are listed in support descending order (in the form of *sequence* : *support*) as below,

$$CS = \{(a) : 5, (c) : 4, (a)(a) : 3, (a)(c) : 3, (a)(b) : 3, (c)(a) : 3\} ;$$
$$FS = \{(a) : 5, (c) : 4, (a)(a) : 3, (a)(c) : 3, (a)(b) : 3, (c)(a) : 3, (b) : 3\} .$$

CS has the exact same information as FS, but includes much fewer patterns.

Table 1. A sample sequence database D

Identifier	Sequence
1	$\langle(ce)(a)(af)(c)\rangle$
2	$\langle(a)(a)(b)(d)\rangle$
3	$\langle(a)(bf)(b)(c)\rangle$
4	$\langle(b)(c)(a)(b)\rangle$
5	$\langle(a)(c)(a)(be)\rangle$

2.2 Related Models of Mining Sequential Pattern

First is an episode discovery approach presented by Mannila et al in [28]. Sequences of events describing the behavior and actions of users or systems can be collected in several domains. An episode is a collection of events that occur relatively close to each other in a given partial order. Once such episodes are known, one can produce rules for describing or predicting the behavior of the sequence. The problem of this model is to find all patterns that occur in some user-specified percentage of windows through moving a time window across the input sequence. It could provide such information like "events of type b, c, and d occur together in 8% of windows of 50 time units" or "event of type b is followed by event of type d 20 times in the event sequence".

Second is the generalized sequential patterns also proposed by Srikant and Agarwal in [5], motivated by real applications such as in a book club. Time constraints that specify a minimum and/or maximum time period between adjacent elements in a pattern are incorporated in the original model and the items to be presented in a set of transactions whose transaction-times are within a user-specified time window are allowed. Given a user-defined taxonomy on items, sequential patterns are also allowed to include items across all levels of the taxonomy. The problem of this model is to find all sequences whose support is greater than the user-specified minimum support, given a database D of data-sequences, a taxonomy T, user-specified min-gap and max-gap time constraints, and a user-specified sliding-window size.

Third is a universal formulation of sequential patterns proposed in [29], which can unify and generalize the above formulations. There are two novel concepts in this universal formulation. One is the directed acyclic graph representation of the structural and timing constraints of sequential patterns. The other is that this approach supplies several different ways in which support of a pattern can be defined, each of which can be suitable in specific applications, depending on the user's perception. By choosing specific combinations of structural constraints, timing constraints, and support counting methods, this formulation can be made identical to the above formulations. A sequential pattern is said to be interesting if it occurs *enough* number of times (support threshold) satisfying the given timing constraint (ms, ws, xg, ng), where ms, ws, xg, ng represent maximum span, event-set window size,

maximum gap, minimum gap, respectively. The problem of this model is to find all interesting sequential patterns, given timing constraint (ms, ws, xg, ng) and support threshold.

In sum up, investigation of models is the first step to discover useful information from database, which helps to support more efficient decision making. It is an important and interesting problem still worthy of study in the future.

3 Analysis of Sequential Pattern Mining Algorithms

Due to its extensive applications, lots of sequential mining algorithms were developed to increase efficiency and effectiveness. They can be categorized into the following classes: BFS(Breadth-first Search)-based method, DFS(Depth-first Search)-based method, closed sequential pattern based method, parallel-based method, incremental-based method and constraint-based algorithm. Here we study several representative algorithms in those classes respectively.

3.1 BFS-Based Method

AprioriAll, AprioriSome and DynamicSome, these three algorithms are first proposed in the pioneering work of sequential pattern mining by Agrawal and Srikant [1]. The latter two algorithms were developed to discover only maximal sequential patterns. The first algorithm, AprioriAll, finds all patterns. It is regarded as a three-phase algorithm in brief. It first finds all frequent itemsets using Apriori, transforms the database so that each transaction is replaced by the set of all frequent itemsets contained in the transaction, and then makes multiple passes over the database to generate candidates, count the supports of candidates, and to discover sequential patterns. Its performance was shown better than or comparable to the other two algorithms. However, this approach nearly doubles the disk space requirement which could be prohibitive for large databases.

The GSP algorithm, proposed in [2], performs like the AprioriAll algorithm, but it does not need find all the frequent itemsets first. In addition, it allows for (1) time-gap constraints, placing bounds on the time separation between adjacent elements in a pattern, (2) sliding time windows, permitting the items in an element of a pattern to span a set of transactions within a user-specified time window, (3) item taxonomies, given a user-defined taxonomy (is-a hierarchy), enabling the discovery of patterns across different levels of a user-defined taxonomy. GSP is also designed to discover these generalized sequential patterns. It makes multiple passes over the database and finds out frequent k-sequences at the kth database scanning. In each pass, every data sequence is examined to update the support counts of the candidates contained in this sequence. Initially, each item is a candidate 1-sequence for the first pass. Frequent 1-sequences are determined after checking all the data sequences in the database. In succeeding passes, frequent $(k-1)$-sequences

are self-joined to generate candidate k-sequences. Again, the supports of these candidate sequences are counted by examining all data sequences, and then those candidates having minimum supports become frequent sequences. This process terminates when there is no candidate sequence left. Empirical evaluation indicates that GSP is much faster than the AprioriAll algorithm by up to 20 times. However, because of the sliding window, minimum and maximum time gaps, it needs backtracking to check if the data sequence s contains each candidate sequence stored in the leaf, which leads to the high computational cost. In addition, its I/O cost may be very high since the number of I/O passes required is determined by the length of the longest frequent sequences.

MFS, proposed in [30], is a modified version of GSP. It tries to reduce the I/O cost needed by GSP. With GSP, every database scan discovers frequent sequences of the same length. MFS, on the other hand, takes a successive refinement approach. It first computes a rough estimate of the set of all frequent sequences as a suggested frequent sequence set, makes use of it and generalizes the candidate generation function of GSP to maintain the set of maximal frequent sequences known so far. Then, longer sequences can be generated and counted early, which is the major source of efficiency improvement of MFS over GSP. Experiment results show that MFS saves I/O cost significantly compared with GSP.

SPADE, proposed in [6], uses the observation that the subsequence relation induces a lattice which is *downward closed* on the support, i.e., if β is frequent, then all subsequences $\alpha\beta$ are also frequent. It decomposes the original lattice into smaller sub-lattices, so that each sublattice can be processed entirely in main-memory using a breadth-first or depth-first search (SPADE also belongs to DFS-based method) for frequent sequences. Starting with the frequent single items, during each step the frequent sequences of the previous level are extended by one more item. Before computing the support of a new sequence, a pruning step ensures that all its subsequences are also frequent, greatly reducing the search space. The experimental results show that SPADE is about twice as fast as GSP. In addition, if we do not count the cost of computing frequent 2-item sequences, SPADE outperforms GSP by an order of magnitude in most cases. The reason is that SPADE uses a more efficient support counting method based on the idlist structure. Furthermore, SPADE only scans the original database twice to generate frequent 1-item sequences and 2-item sequences respectively, and the remaining operations are solely performed on the idlist of each sequence, which keeps shrinking during the mining process and is much smaller than the original database. SPADE also shows a linear scalability with respect to the number of sequences.

3.2 DFS-Based Method

In [31], Han et al. proposed a projection-based algorithm called FreeSpan, which aims at reducing the generation of candidate subsequences. Its general idea is to use frequent items to recursively project sequence databases into

a set of smaller projected databases based on the currently mined frequent sets, and grow subsequence fragments in each projected database respectively. This process partitions both the data and the set of frequent patterns to be tested, and confines each test being conducted to the corresponding smaller projected database. FreeSpan only needs to scan the original database three times, independent of the maximal length of the sequence. Performance study shows that FreeSpan mines the complete set of patterns and is efficient and runs considerably faster than the GSP algorithm. The major cost of FreeSpan is to deal with projected databases. Moreover, since a length-k subsequence may grow at any position, the search for length-$(k+1)$ candidate sequences will need to check every possible combination, which is quite costly.

To solve this problem existing in FreeSpan, in [7], Pei et al. proposed another projection based algorithm called PrefixSpan. Its general idea is to examine only the prefix subsequences and project only their corresponding postfix subsequences into projected databases, instead of projecting sequence databases by considering all the possible occurrences of frequent subsequences. In each projected database, sequential patterns are grown by exploring only local frequent patterns. PrefixSpan mines the complete set of patterns and is efficient and runs considerably faster than both GSP algorithm and FreeSpan. However, similar to FreeSpan, the major cost of PrefixSpan is also the construction of projected databases. In the worse case, PrefixSpan needs to construct a projected database for every sequential pattern. If there are a large number of sequential patterns, the cost is non-trivial.

In [32], Sequential PAttern Mining using a bitmap representation (SPAM) was proposed by J. Ayres, et al. Based on a lexicographic tree of sequences, SPAM utilizes a depth-first traversal of the search space combined with a vertical bitmap representation to store each sequence, which allows for efficient support counting as well as significant bitmap compression. In addition, various pruning mechanisms are implemented to reduce the search space. SPAM is especially efficient when the sequential patterns in the database are very long. Moreover, a salient feature of this algorithm is that it incrementally outputs new frequent itemsets in an online fashion. Experimental results demonstrate that this algorithm outperforms SPADE and PrefixSpan on large datasets by over an order of magnitude. However, SPAM assumes that the entire database (and all data structures used for the algorithm) completely fit into main memory which is not suitable for mining sequential pattern from large databases and it consumes more space in comparison with SPADE and PrefixSpan.

3.3 Closed Sequential Pattern Based Method

Previous sequential pattern mining algorithms mine the full set of frequent subsequences satisfying a minimum support threshold in a sequence database. However, since a frequent long sequence contains a combinatorial number of frequent subsequences, such mining will generate an explosive number of frequent subsequences for long patterns, which is prohibitively expensive in

both time and space. It is proved that for mining frequent patterns (for both itemsets and sequences), one should not mine *all* frequent patterns but the *closed* ones since the latter leads to not only more compact yet complete result set but also better efficiency, which can greatly reduces the number of frequent subsequences [4, 11].

CloSpan is such an algorithm for mining closed sequential patterns [4]. It divides the mining process into two stages. In the first stage, a candidate set is generated. The second stage helps eliminate non-closed sequences. CloSpan develops several efficient search space pruning methods and it is hash-based algorithm which can efficiently execute the search space optimization with negligible cost. The performance of CloSpan shows that it not only generates a complete closed subsequence set which is substantially smaller than that generated by PrefixSpan, but also runs much faster. However, it follows a *candidate maintenance-and-test* paradigm and results in a rather poor scalability in the number of frequent closed patterns because a large number of frequent closed patterns (or just candidates) will occupy much memory and lead to a large search space for the closure checking of new patterns, which is usually the case when the support threshold is low or the patterns become long.

BIDE is a more efficient algorithm to mine closed sequential patterns [11]. It avoids the curse of the *candidate maintenance-and-test* paradigm, prunes the search space more deeply and checks the pattern closure in a more efficient way while consuming much less memory in contrast to the previously developed closed pattern mining algorithms. It does not need to maintain the set of historic closed patterns, thus it scales very well in the number of frequent closed patterns. BIDE adopts a strict depth-first search order and can output the frequent closed patterns in an online fashion. A thorough performance study shows that BIDE consumes order(s) of magnitude less memory and runs over an order of magnitude faster than the previously developed frequent (closed) sequence mining algorithms, especially when the support is low. It also has linear scalability in terms of the number of sequences in the database. However, like other closed sequence mining algorithms, it will lose to some all-frequent-sequence mining algorithms with a high support threshold.

3.4 Parallel-Based Method

The most time consuming operation in the discovery process of sequential patterns is the computation of the frequency of the occurrences of interesting subsequences in the sequence database. However, the number of sequential patterns grows exponentially and the task of finding all sequential patterns requires a lot of computational resources, which make it an ideal candidate for parallel processing.

Three parallel algorithms, NPSPM, SPSPM, HPSPM, based on GSP for mining sequential patterns on a shared-nothing environment were presented in [8]. All three approaches partition the datasets into equal sized blocks

among the nodes. In NPSPM, the candidate sequences are replicated on all the processors, and each processor gathers local support using its local database block. A reduction is performed after each iteration to get the global supports. Since NPSPM replicates the entire candidate set on each node, it can run into memory overflow problems for large databases. SPSPM partitions the candidate set into equal-sized blocks and assigns each block to a separate processor. While SPSPM utilizes the aggregate memory of the system, it suffers from excessive communication, since each processor's local database has to broadcast to all other processors to get global support. HPSPM uses a more intelligent strategy to partition the candidate sequences among the nodes using hash function, which eliminates the customer transaction data broadcasting and reduces the comparison workload. Among three algorithms HPSPM was shown to be the best approach through the experiments on an IBM SP2 distributed memory machine. However, the main limitation of all these parallel algorithms is that they make repeated passes over the disk-resident database partition, incurring high I/O overheads. Furthermore, the schemes involve exchanging the remote database partitions during each iteration, resulting high communication and synchronization overhead. They also use complicated hash structures, which entail additional overhead in maintenance and search, and typically also have poor cache locality [9].

pSPADE is a parallel algorithm based on SPADE for fast discovery of frequent sequences in large databases, targeting shared-memory systems [9]. It decomposes the original search space into small suffix-based classes. Each class can be solved in main-memory using simple join operations and efficient search techniques, which not only minimizes I/O costs by reducing database scans, but also computational costs. Further each class can be solved independently on each processor requiring no synchronization. It has good locality and little false sharing. Experiments on a 12 processor SGI Origin 2000 shared memory system show that pSPADE delivers good speedup and has excellent scale-up properties. However, like SPADE, the limitation of pSPADE is that it works on the assumption that each class and its intermediate idlists fit in main memory, which require lots of memory.

DPF, TPF are two different parallel algorithms for finding sequential patterns on distributed-memory parallel computers [10]. DPF decomposes the computation by exploiting data parallelism, whereas TPF utilizes task parallelism. The feature of TPF is the development of a static task decomposition scheme that uses a bipartite graph partitioning algorithm to simultaneously balance the computations and at the same time reduce the data sharing overheads, by minimizing the portions of the database that needs to be shared by different processors. Experiments on the 32-processor IBM SP2 parallel computer show that they incur small communication overheads, achieve good speedups, and can effective utilize the different processors, and that TPF outperformed DPF. However, as number of processors increased, the accuracy of estimated work-load decreased and the computation became increasingly un-balanced.

DTPF is an improved parallel algorithm of TPF, which uses task parallelism along with dynamic load balancing scheme that minimizes idle time in case when distributed workload is unbalanced [33]. Experiments on the 32-processor IBM SP2 parallel computer show that it is capable of achieving good speedups, substantially reducing the amount of the required work to find sequential patterns in large databases, and it outperforms static load balancing scheme algorithm, TPF.

3.5 Incremental-Based Method

The above studies all assume the database is static, and even a small change in the database will require the algorithms to run again to get the updated frequent sequences since previous sequential patterns would become irrelevant and new sequential patterns might appear. In practice, the content of a database changes over time and thus there is a need for efficient algorithms to update, maintain and manage the information discovered. If each time we have to rerun the mining algorithms from scratch, it will be very inefficient or infeasible. Incremental algorithm should be developed for sequential pattern mining so that mining can be adapted to frequent and incremental database updates, including both insertions and deletions. There are two cases in developing incremental algorithm: (1) Whole sequences are inserted into and/or removed from the old database called as Sequence Model; (2) Sequences in the old database are updated by appending new transactions at the end called as Transaction Model. The two models can model each other. Algorithms designed for one model can also work under the other model.

A work for incremental sequential pattern updating based on *SuffixTree* techniques was proposed in [12]. The structure used in that context acquires the data and builds up the frequent sequences in one scan by means of a *SuffixTree*. This method is thus very appropriate to an incremental sequence extraction, because it only has to continue the data reading after the update. The limitations are the complexity in space of this algorithm depends on the size of the database and the *SuffixTree* is very expensive for dynamic strings because of the sensitivity of the position to the update operation. Then, a modified work of that on incremental sequential pattern updating was proposed in [13]. The approach uses a dynamic *SuffixTree* structure, in which substrings are referenced by addresses rather than positions, for incremental mining in a single long sequence. The address reference restricts the impact of updates to a small part of the dynamic *SuffixTree*, making efficient update of the dynamic *SuffixTree* possible. Experiments showed that this incremental method performs substantially better than the non-incremental one for large and dynamic databases. However, those two algorithms only focus on incremental mining in a single long sequence.

FASTUP, proposed in [34], is in effect an enhanced GSP with improvement on candidate generation and support counting. It takes into account the previous mining result before generating and validating candidates using the

generating-pruning method. Experiments show that the performance of this algorithm could be much faster than previous algorithms for the maintenance of sequential patterns. However, it suffers the same limitations as GSP.

Reference [14] developed an incremental mining algorithm ISM based on SPADE by maintaining a sequence lattice of an old database. The sequence lattice includes all the frequent sequences and all the sequences in the negative border, which is the collection of all sequences that are not frequent but both of whose generating subsequences are frequent. Compared with SPADE, the experiment results show that ISM is an improvement in execution time by up to several orders of magnitude in practice, both for handling increments to the database, as well as for handling interactive queries. However, maintaining negative border, the number of which can be very huge, is memory consuming and not well adapted for large databases. In addition, sequences in the negative border may be unlikely to become frequent in the updated database if they have low support. Moreover, ISM can only deal with the case of insertion.

Reference [16] developed two algorithms, GSP+ and MFS+, for incremental mining sequential patterns when sequences are inserted into or deleted from the original database: one based on GSP and the other based on MFS. These two algorithms can efficiently compute the updated set of frequent sequences given the set of frequent sequences obtained from mining the old database. Experiments show that GSP+ and MFS+ effectively reduce the CPU costs of their counterparts with only a small or even negative expense on I/O cost.

Reference [35] developed another incremental mining algorithm, ISE, using candidate generate-and-test approach, namely, using information collected during an earlier mining process to cut down the cost of finding new sequential patterns in the updated database. The main new feature of it is that the set of candidate sequences to be tested is substantially reduced. Furthermore, it incorporates some optimization techniques for improving the efficiency. Empirical evaluations show that the algorithm performs significantly faster than the approach, GSP, of mining the whole updated database from scratch. The limitation of this algorithm is the candidate set can be very huge, which makes the test-phase very slow and its level-wise working manner requires multiple scans of the whole database. This is very costly, especially when the sequences are long.

IncSP is another efficient incremental updating algorithm for up-to-date maintenance of sequential patterns after a nontrivial number of data sequences are appended to the sequence database [36]. Assume that the minimum support keeps the same. It utilizes the knowledge of previously computed frequent sequences, merges data sequences implicitly, prunes candidates early, and separately counts supports with respect to the original database and the newly appended database. Implicit merging ensures that IncSP employs correctly combined data sequences while preserving previous knowledge useful for incremental updating. Candidate pruning after updating pattern supports against the increment database further accelerates the whole process, since fewer but more promising candidates are generated by just checking counts

in the increment database. Eventually, efficient support counting of promising candidates over the original database accomplishes the discovery of new patterns. IncSP both updates the supports of existing patterns and finds out new patterns for the updated database. The simulation performed shows that IncSP is several times faster than re-mining using the GSP algorithm, with respect to various data characteristics or data combinations. IncSP outperforms GSP with regard to different ratios of the increment database to the original database except when the increment database becomes larger than the original database.

IncSpan is developed in [37] for incremental mining over multiple database increments. Two novel ideas are introduced in the algorithm development. First is maintaining a set of "*almost frequent*" sequences as the candidates in the updated database, which has several nice properties and leads to efficient techniques. Second is that two optimization techniques, *reverse pattern matching* and *shared projection*, are designed to improve the performance. Reverse pattern matching is used for matching a sequential pattern in a sequence. Since the appended transactions are at the end of a sequence, reverse pattern matching can prune additional search space. Shared projection is designed to reduce the number of database projections for some sequences which share a common prefix. Performance study shows that IncSpan outperforms ISM and PrefixSpan on incrementally updated databases by a wide margin.

3.6 Constraint-Based Algorithm

Recent work has highlighted the importance of the paradigm of constraint-based mining. Not only the paradigm allows users to express their focus in mining, but also allows many kinds of constraints to be pushed deep inside mining, confining the search for patterns only to those of interest to the user, and therefore, improving performance. Constraint-based algorithms are close related to user-specified constraints because of the arising problem, namely, how to push kinds of constraints deep inside mining in order to improve performance.

The use of Regular Expressions (REs) was proposed in [18] as a flexible constraint specification tool that enables user-controlled focus to be incorporated into the pattern mining process. A family of novel algorithms, SPIRIT, was developed for mining frequent sequential patterns that also satisfy user-specified RE constraints. The main distinguishing factor among the proposed schemes is the degree to which the RE constraints are enforced to prune the search space of patterns during computation. The SPIRIT algorithms are illustrative of the tradeoffs that arise when constraints that do not subscribe to nice properties (like anti-monotonicity) are integrated into the mining process. Experimental results clearly validate the effectiveness of the approach, showing that speedups of more than an order of magnitude are possible when RE constraints are pushed deep inside the mining process and also illustrates the versatility of REs as a user-level tool for focusing on interesting patterns.

cSPADE was proposed in [38] for discovering the set of all frequent sequences with the following constraints: length and width restrictions, minimum and maximum gap between sequence elements, time window of occurrence of the whole sequence, item constraints for including or excluding certain items, and finding sequences distinctive of at least one class, i.e., a special attribute-value pair, that we are interested in predicting. The two main strengths of cSPADE are that it delivers performance far superior to existing approaches to constrained sequences, and that it incorporates the constraints with relative ease.

Prefix-growth is also one of constraint-based algorithms for mining sequential patterns developed in [19] to push prefix-monotone constraints, which covers many commonly used constraints such as all monotonic, anti-monotonic constraints and regular expression, deep into the mining process. Moreover, some tough constraints, like those involving aggregate $avg()$ and $sum()$, can also be pushed deep into prefix-growth with some minor extensions. Experimental results and performance study show that prefix-growth is efficient and scalable in mining large databases with various constraints compared with SPIRIT.

Moreover, there are many other kinds of algorithms for mining sequential patterns like DSG (Direct sequential pattern generation), a graph-based algorithm, proposed in [39]. Though the disk I/O cost of DSG is very low because it scans database only once, the related information may not fit in the memory when the size of the database is large.

All in all, the existing algorithms for mining sequential patterns depend heavily on massive computation that might cause high dependency on the memory size or repeated I/O scans for the data sets. Though they are very efficient, they are not sufficient for extremely large datasets and new solutions, that do not depend on repeated I/O scans and less reliant on memory size, still have to be found.

4 Theoretical Foundation of Sequential Pattern Mining

Previous work on sequential pattern discovery was mainly focused on studying scalable algorithms while less on its theoretical foundations, which is also important and makes it possible to use the existing theories or methods to support more effective sequential pattern mining tasks. In the following, we build the relation between the closed sequential patterns and the fixed point by using the theory of formal concept analysis [40, 41], which can serve as a theoretical foundation of sequential patterns [42].

Definition 1. *Reference [40] A triple (G, M, I) is called a context if G and M are sets and $I \subseteq G \times M$ is a binary relation between G and M. We call the elements of G objects, those of M attributes, and I the incidence of the context (G, M, I).*

For the object g and the attribute m, $(g, m) \in I$ or more commonly, gIm implies that 'the object g possesses the attribute m'.

Definition 2. *Reference [40] Let (G, M, R) be a context, then the following mappings are Galois connections between $P(G)$ and $P(M)$:*

$$s : G \mapsto M, s(X) = \{m \in M \,|(\forall g \in X)gRm\} ,$$

$$t : M \mapsto G, t(Y) = \{\, g \in G|\, (\forall m \in Y)gRm\} ,$$

where $P(G)$ and $P(M)$ are the power sets of G and M, respectively.

Obviously, $s \circ t$ and $t \circ s$ are closed operators, also call them as Galois closed operators. Let the identifier set ID, the sequential set D of a sequence database be G and M of a context (G, M, R), a binary relation between U and D be R, then (U, D, R) becomes a context. Their Galois connections are as follows:

$$t : D \mapsto U, t(\alpha) = \{\, g \in U|\, (\forall m \in \alpha)gRm\}$$

$$s : U \mapsto D, s(X) = \{\, m \in D|\, (\forall g \in X)gRm\}$$

Then, $t(\alpha)$ denotes all the id set that includes a sequence α. Moreover, $s \circ t$ and $t \circ s$ are closed operators, also call them as Galois closed operators.

Theorem 1. *$\{\alpha \in P(D) \,|s \circ t(\alpha) = \alpha\}$ is the set of all closed sequential patterns of sequence database D.*

Proof. (Sufficiency) Suppose that a sequence α, satisfying $s \circ t(\alpha) = \alpha$, is not a closed sequential pattern. Then there exists a sequence β, satisfying $\alpha \subset \beta$ and $support(\alpha) = support(\beta)$. Namely, $t(\alpha) = t(\beta)$ and thus $s \circ t(\beta) \supseteq \beta$. It is concluded that $\alpha = s \circ t(\alpha) = s \circ t(\beta) \supseteq \beta$, which contradicts that $\alpha \subset \beta$. Therefore, α is a closed sequential pattern.

(Necessity) Suppose that a sequence α is a closed sequential pattern of D. Since $s \circ t$ is a closed operator, $s \circ t(\alpha) \supseteq \alpha$. If $s \circ t(\alpha) \supset \alpha$, according to the definition of s and t, every sequence contains α also contains $s \circ t(\alpha)$. Thus, $support(s \circ t(\alpha)) = support(\alpha)$, which contradicts the assumption that α is a closed sequential pattern. Therefore, we have $s \circ t(\alpha) = \alpha$. □

Definition 3. *Reference [43] Let P be a partial order set, Φ:$P \to P$ is a mapping, $a \in P$. If $\Phi(a) = a$, then call a is a fixed point of Φ.*

Then, every sequence in the set $\{\alpha \subseteq D \,|s \circ t(\alpha) = \alpha\}$ is a fixed point of the mapping $s \circ t$. To mine all frequent closed sequential patterns is equal to find all fixed points of $s \circ t$. The existence of fixed point of the mapping is confirmed by the following theorems.

Theorem 2. *Reference [43] Let P be a partial order set, Φ: $P \to P$ is a mapping, satisfy that for every $a \in P, a \leq \Phi\ (a)$, then Φ has fixed points.*

Theorem 3. [43] *Let P be a partial order set, Φ:$P \to P$ is an order-preserving mapping, then Φ has fixed points and the minimum fixed point.*

Theorem 4. *$s \circ t : P(D) \to P(D)$ has fixed points.*

Proof. It is obvious that $P(D)$ is a partial order set. Since $s \circ t : P(D) \to P(D)$ is a closed operator. Then, for every $\alpha \in P(D)$, $\alpha \subseteq s \circ t(\alpha)$ and $s \circ t$ is a order-preserving mapping. Therefore, there exist fixed points of $s \circ t$.

Since $(P(D), \subseteq)$ is a complete lattice and closed operator is order-preserving mapping, then a concrete fixed point(namely, closed sequential pattern) can be obtained by the fixed point theorem proposed by Knaster and Tarski [43]. □

Theorem 5. *Reference [43] Let P be a complete lattice, Φ: $P \to P$ is order-preserving mapping, then $\vee\{a \in P \,|\, a \leq \Phi(a)\}$ is a fixed point of Φ.*

Theorem 6. *$(\{\alpha \in P(D) \,|\, s \circ t(\alpha) = \alpha\}, \subseteq)$ constitues a join semi-lattice, called as fixed point semi-lattice.*

Proof. It is obvious. □

Because the closed sequential patterns keep all the support information of all sequential patterns of sequence database, fixed point semi-lattice also keeps them. Therefore, to mine all frequent closed sequential patterns is equal to establish fixed point semi-lattice and mine all points on it that satisfy support constraint.

Example 2. Frequent fixed-point semi-lattice of Table 1 is as follows (Fig. 1). Every node's support in this lattice is above the minimum support threshold.

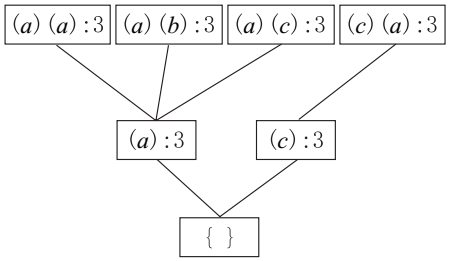

Fig. 1. Frequent fixed-point semi-lattice

To sum up, much work has been done to efficiently discovery sequential patterns while less work on its theoretical foundations. Based on the Galois closed operator, identifier set, sequential set in sequential database together with their binary relation constituted a context. The relation between the fixed point and closed sequential pattern was established. To mine all frequent

closed sequential patterns is equal to build fixed point semi-lattice and mine all points on it that satisfy support constraint which serves as a theoretical foundation of sequential patterns and makes it possible to use the existing fixed point and lattice theories to support more effective sequential pattern mining tasks.

5 Applications of Sequential Pattern Mining

In the daily and scientific life, sequential data are available and used everywhere. Since discovering interesting patterns can benefit us by predicting coming activities, interpreting certain phenomena, extracting outstanding similarities and differences for close attention, etc. Methods for mining sequential patterns have successfully applied in many domains as follows.

- Retail industry: analysis of customer buying behavior.
- Medical treatment: discovering patterns in histories of medical records to improve level of diagnosis.
- DNA sequences and gene structures: discovery of motifs and tandem repeats in DNA sequences.
- Telecommunication: finding network alarm patterns, telephone calling patterns.
- Administration: identifying plan failures.
- Web service: discovering user access patterns.
- Information security: analysis of user's behavior.
- Natural disasters: earthquakes forecasting.
- Science & engineering processes: study of engineering & scientific processes such as experiment runs, satellite data streams, weather data.
- Stocks and markets: stock prices trend.

Following is an example which is a successful application of sequential pattern mining. PLANMINE, an automatic mining method that discovers event sequences causing failures in plans, was presented in [2]. Novel pruning techniques to extract the set of the most predictive rules from highly structured plan databases were developed. These pruning strategies reduced the size of the rule set by three orders of magnitude. PLANMINE has been fully integrated into two real-world planning systems. The rules discovered by PLANMINE were extremely useful for understanding and improving plans, as well as for building monitors that raise alarms before failures happen.

All in all, several cases should be carefully studied during the process of applications of sequential pattern mining: (1) Developing operations for data cleaning; (2) Integration of the algorithm into the real-world system; (3) Devising approaches to reduce the size of the rule set; (4) Visualization should also be carefully considered.

6 Conclusions

Sequential pattern mining is one of the most popular pattern-discovery approaches in the field of knowledge discovery and data mining. In this chapter, we conduct a systematic study on models and algorithms in sequential pattern analysis, build the relation between the closed sequential patterns and the fixed point and discuss its application domains. Despite recent advance in the problem of sequential pattern mining, several problems still need serious and immediate attention.

- Developing more efficient and scalable methods for mining sequential patterns, including incremental mining of closed sequential patterns, incorporating user-specified constraints in the mining of closed sequential patterns, developing methods to mine sequential patterns with other more complicated constraints, devising sampling-based methods and random access disk-based approaches like Inverted Matrix for efficient discovery of frequent itemsets [44, 45], etc.
- Proceeding to explore its application domains, including development of application-specific data mining system, invisible data mining (mining as built-in function) and integration of existing algorithms for other complicated structured patterns, etc.
- To continue studying theoretical foundations of sequential pattern mining like Codd's relational model [46] in order for us to apply them to the development of more efficient mining algorithms and methods.
- To establish a benchmark to evaluate sequential pattern mining algorithms like FIMI workshop for the problem of frequent itemset mining [47] in order to generate a very healthy and critical discussion on the state-of-affairs in sequential pattern mining implementations.
- To integrate efficient algorithms with database management systems, data warehouse systems, and Web database systems, etc. which can maximally benefit end-users.
- Development of techniques that incorporate privacy concerns in sequential pattern mining since data mining, with its promise to efficiently discover valuable, non obvious information from large databases, is particularly vulnerable to misuse [48, 49].

References

1. Agrawal R, Srikant R (1995) Mining sequential patterns. In Proc of the 11th Int Conf on Data Eng. Mar. 1995, Taipei, Taiwan. 3–14 103, 104, 105, 107
2. Zaki MJ, Lesh N, Ogihara M (2000) PLANMINE: Sequence mining for plan failures. Artificial Intelligence Review, special issue on the Application of Data Mining 14(6): 421–446 103, 107, 118
3. Wu PH, Peng WC, Chen MS (2001) Mining sequential alarm patterns in a telecommunication database. In Proc of VLDB-01 Workshop on Databases in Telecommunications. Sept. 2001, Roma, Italy. 37–51 103

4. Yan X, Han J, Afshar R (2003) CloSpan: Mining Closed Sequential Patterns in Large Databases. In SDM'03. May 2003, San Francisco, CA. 166–177 104, 110
5. Srikant R, Agrawal R (1996) Mining sequential patterns: generalizations and performance improvements. In Proc of the 5th Int Conf on Extending Database Technology. Mar. 1996, Avignon, France. 3–17 104, 106
6. Zaki MJ (2001) An eficient agorithm for mining fequent sequences. Machine Learning J 42(1/2): 31–60 104, 108
7. Pei J, Han J, Pinto H, Chen Q, Dayal U, Hsu MC (2001) PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In Proc of 2001 Int Conf on Data Eng. Apr. 2001, Heidelberg, Germany. 215–224 104, 109
8. Shintani T, Kitsuregawa M (1998) Mining algorithms for sequential patterns in parallel: Hash based approach. In Proc of the Second Pacific-Asia Conf on Know Discovery and Data mining. Apr. 1998, Merburn, Australia. 283–294 104, 110
9. Zaki MJ (2001) Parallel sequence mining on shared-memory machines. J of Parallel and Distributed Computing 61(3): 401–426 104, 111
10. Guralnik V, Garg N, Karypis G (2001) Parallel tree projection algorithm for sequence mining. In Sakellariou R Keane J Gurd J Freeman L (eds) Proc of 7th European Conf on Parallel Computing. 310–320. Springer, Berlin Heidelberg New York 104, 111
11. Wang J, Han J (2004) BIDE: Efficient mining of frequent closed sequences. In Proc. of 2004 Int. Conf. on Data Eng. Apr. 2004, Boston, MA. 79–90 104, 110
12. Wang K, Tan J (1996) Incremental discovery of sequential patterns. In Proc of Workshop on Research Issues on Data Mining and Know Discovery. June 1996, Montreal, Canada. 95–102 104, 112
13. Wang K (1997) Discovering patterns from large and dynamic sequential data. J of Intelligent Information Systems 9(1): 33–56 104, 112
14. Parthasarathy S, Zaki MJ, Ogihara M, Dwarkadas S (1999) Incremental and interactive sequence mining. In Proc of the 8th Int Conf on Information and Know Management. Nov. 1999, Kansas, Missouri, USA. 251–258 104, 113
15. Lee CH, Lin CR, Chen MS (2001) Sliding-Window filtering: An efficient algorithm for incremental mining. In Proc of the ACM 10th Int Conf on Information and Know Management. Oct. 2001, Atlanta, Georgia. 263–270 104
16. Zhang M, Kao B, Cheung D, Yip CL (2002) Efficient algorithms for incremental update of frequent sequences. In Proc of the 6th Pacific-Asia Conf on Know Discovery and Data Mining. May, 2002, Taipei, Taiwan. 186–197 104, 113
17. Masseglia F, Poncelet P, Teisseire M (2003) Incremental mining of sequential patterns in large databases. Data Know. Eng 46: 97–121 104
18. Garofalakis MN, Rastogi R, Shim K (1999) SPIRIT: Sequential pattern mining with regular expression constraints. In Proc of the 25th Int Conf on Very Large Data Bases. Sept. 1999, Edinburgh, Scotland. 223–234 104, 114
19. Pei J, Han J, Wang W (2002) Mining sequential patterns with constraints in large databases. In Proc of the 11th Int Conf on Information and Know Management. Nov. 2002, McLean, VA. 18–25 104, 115
20. Pei J, Han J (2002) Constrained frequent pattern mining: A pattern-growth view. SIGKDD Explorations 4(1): 31–39 104
21. Toroslu IH, Kantarcioglu M (2001) Mining cyclically repeated patterns. In: Kambayashi Y, Winiwarter W, Arikawa M (eds): DaWaK 2001, LNCS 2114. 83–92 104

22. Kum HC, Pei J, Wang W, Duncan D (2003) ApproxMAP: Approximate mining of consensus sequential patterns. In Proc of the 2003 SIAM Int Conf on Data Mining. May 2003, San Francisco, CA. 311–315 104
23. Pinto H, Han J, Pei J, Wang K, Chen Q, Dayal U (2001) Multi-dimensional sequential pattern mining. In Proc of the 10th Int Conf on Information and Know Management. Nov. 2001, Atlanta, Georgia. 81–88 104
24. Tzvetkov P, Yan X, Han J (2003) TSP: Mining top-k closed sequential patterns. In Proc. 2003 Int. Conf. on Data Mining. Nov. 2003, Melbourne, FL. 347–354 104
25. Afshar R (2001) Mining frequent Max and closed patterns. MA thesis, Simon Fraser University, Canada 104
26. Yang J, Yu PS, Wang W, Han J (2002) Mining long sequential patterns in a noisy environment. In SIGMOD'02. June 2002, Madison, WI. 406–417 104
27. Chen YL, Chen SS, Hsu PY (2003) Mining hybrid sequential patterns and sequential rules, Information Systems 27: 345–362 104
28. Mannila H, Toivonen H, Verkamo AI (1995) Discovering frequent episodes in sequences. In Proc of the 1st Int Conf on Know Discovery and Data Mining. Aug. 1995, Montreal, Canada. 210–215 106
29. Joshi MV, Karypis G, Kumar V (1999) Universal formulation of sequential patterns. Technical Report Under Preparation. Department of Computer Science, University of Minnesota, Minneapolis 106
30. Zhang M, Kao B, Yip CL, Cheung D (2001) A GSP-based efficient algorithm for mining frequent sequences. In Proc. of IC-AI'2001. June 2001, Las Vegas, Nevada, USA 108
31. Han J, Pei J, Mortazavi-Asl B, Chen Q, Dayal U, Hsu MC (2000) FreeSpan: Frequent pattern-projected sequential pattern mining. In Proc of the 6th ACM SIGKDD int conf on Know discovery and data mining. Aug. 2000, Boston, MA. 355–359 108
32. Ayres J, Gehrke JE, Yiu T, Flannick J (2002) Sequential pattern mining using bitmaps. In Proc of the 8th ACM SIGKDD Int Conf on Know Discovery and Data Mining. July 2002, Edmonton, Alberta, Canada. 429–435 109
33. Guralnik V, Karypis G (2001) Dynamic load balancing algorithms for sequence mining. Technical Report 00-056, Department of Computer Science, University of Minnesota 112
34. Lin MY, Lee SY (1998) Incremental update on sequential patterns in large databases. In Proc of the 10th IEEE Int Conf on Tools with Artificial Intelligence. Nov. 1998, Taipei, Taiwan. 24–31 112
35. Masseglia F, Poncelet, Teisseire M (2003) Incremental mining of sequential patterns in large databases. Data & Know Eng 46: 97–121 113
36. Lin MY, Lee SY (2004) Incremental update on sequential patterns in large databases by implicit merging and efficient counting. Information Systems 29: 385-404 113
37. Cheng H, Yan X, Han J (2004) IncSpan: Incremental mining of sequential patterns in large database. In Proc. 2004 Int. Conf. on Know Discovery and Data Mining. Aug. 2004, Seattle, WA. 527–532 114
38. Zaki MJ (2000) Sequence mining in categorical domains: Incorporating constraints. In 9th Int Conf on Information and Know Management. Nov. 2000, Washington, DC. 422–429 115
39. Yen SJ, Chen ALP (1996) An efficient approach to discovering knowledge from large databases. In Proc of 4th Int Conf on Parallel and Distributed Information Systems. IEEE Computer Society. 8–18 115

40. Ganter B, Wille R (1999) Formal concept analysis: Mathematical foundations. Springer, Berlin Heidelberg, New York 115, 116
41. Pasquier N, Bastide Y, Taouil R, Lakhal L (1999) Efficient mining of association rules using closed itemset lattices. Information Systems 24: 25–46 115
42. Li TR, Yang N, Ma J, Xu Y (2004) Theoretical foundations of sequential patterns. In The World Congress on Intelligent Control and Automation. June 2004, Hangzhou, China. 4241–4244 115
43. Smart DR (1974) Fixed point theorems. Cambridge University Press, Cambridge, UK 116, 117
44. Lee S, Cheung D, Kao B (1998) Is sampling useful in data mining? A case in the maintenance of discovered association rule, Data Mining and Know Discovery 2(3): 233–262 119
45. EI-Hajj M, Zaiane OR (2003) Inverted Matrix: Efficient discovery of frequent items in large datasets in the context of interactive mining, In Proc. of the 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining. Aug. 2003, Washington, DC. 109–118 119
46. Mannila H (2000) Theoretical frameworks of data mining, SIGKDD explorations 1(2): 30–32 119
47. Goethals B, Zaki MJ (2003) Introduction: Advances in frequent itemset mining implementations. In Proc of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations. Nov. 2003, Melbourne, Florida, USA. 1–13 119
48. Agrawal R, Srikant R (2000) Privacy-preserving data mining. In Proc of the ACM SIGMOD Conf on Management of Data. May 2000, Dallas, TX. 439–450 119
49. Zhan Z, Chang LW, Matwin S (2004) Privacy-preserving collaborative sequential pattern mining. In SIAM DM 2004 Workshop on Link Analysis, Counter-Terrorism & Privacy. Apr. 2004, Lake Buena Vista, Florida 119