Master research Internship

Bibliographic report

Negative temporal pattern mining

Domain: Artificial Intelligence - Data Mining

Author:
Aikaterini Tsesmeli

Supervisor:
Thomas Guyet
René Quiniou
EPI LACODAM

Abstract: Mining sequential patterns is one the most important tasks for the research field but it also finds an impact on companies that are interested in mining sequences corresponding to multimedia interaction of customers with their services. In this context, we are interested in mining undesirable situations, that can be defined with the terms of the negative patterns and exploring the field of event prediction in order to find efficient ways to predict the occurrence of such events, like the undesirable phone calls from customers to the company, and finally be able to avoid them in future situations.

Contents

1 Introduction **1**

2 Pattern mining **2**
2.1 Itemset and association rule mining 2
2.2 Sequential pattern mining 4
2.2.1 Mining sequential pattern under constraints 5
2.3 Temporal pattern mining 6

3 Negative sequential analysis **7**
3.1 Negative sequential pattern mining 8

4 Event prediction **11**

5 Conclusion **14**

1 Introduction

More and more companies dispose various ways of communication with their customers, in order to interact with them (web site, phone calls, postal or electronic mail, etc). However, each company, for its own reasons, prefers to avoid that some of these ways will be used by its customers. In this context, our industrial partner, EDF[1], prefers that people who work on the call center will be concentrated in the most complex situations and will not be occupied by simple clients phone calls. So, the most preferable situation for EDF is that its customers do not use phone in order to communicate with the company.

The main motivation of our work is to mine sequences corresponding to multimedia interaction of customers with the services of EDF and finally to predict the occurrence of undesirable events that will help to avoid the use by customers of certain media expensive for the company or not appropriate for some complex requests.

In the general framework of the operational analysis, monitoring systems are searching to detect early an evolution of the supervised system towards undesirable situations and also to indicate actions (inhibiting actions) that could avoid these situations.In this report, we are interested in extracting rules that could predict, the earliest possible, an evolution towards an undesirable situation and indicate actions that could be implemented and help us avoid the occurrence of undesirable events. Such rules are rarely available and difficult to be formalized by the experts of the field. It is necessary to learn them or to discover them automatically from the mass of data available in the monitoring logs. In the interaction data that we will use, we assume that we have two types of interactions, some interactions that will have resulted in undesirable phone calls and some others, in the similar context, for which the sending of an appropriate information has led to avoid a phone call.

Data mining is an unsupervised learning method, such as clustering, or pattern discovery and several methods and algorithms have been designed to "scale" and process large data. We would like to use this type of methods in order to discover and extract sequential patterns which predict accurately the occurrence of undesirable situations. Moreover, we could use patterns that define negative events or sequential patterns that can be predicted at an early stage. With the term "negative patterns" we refer to sequences that contain non-occurring events too. However, it could be interesting to find these non-occuring events, the occurence of which could maybe change a situation from undesirable to desirable. So, in the concept of event prediction, we would also like to discover events that could be implemented in the extracted negative patterns and influence the behavior of the negative events. For example, in the case of communication patterns between EDF and its customers, if we are able to extract the undesirable patterns and define relative negative patterns, it will be useful to implement inhibiting actions in order to change the behavior of undesirable situations like the phone calls.

In the following chapters, we will talk about different aspects of the data mining method, by presenting some state of the art pattern mining algorithms. After that, we will focus on the analysis of negative sequential patterns and finally, we will talk about event prediction and some efficient methods found in the literature.

[1]EDF: french electric utility company.

2 Pattern mining

2.1 Itemset and association rule mining

Association rules mining has received considerable attention by the research community, particularly since the publication of the Apriori algorithm by Agrawal and Srikant [1]. The task concerns discovering correlations between items in a dataset and it is one of the most important data mining techniques. Before continuing with details in association rules mining, we need to give some definitions, supported by an example.

Frequent itemsets The seminal work of Aggrawal and Srikant [1] introduces the canonical example of market basket analysis that was reused by most subsequent research in data mining. Market basket data are illustrated by table 1 which represents a *transaction database* D, where each row is a customer transaction. Each transaction consists of two fields: transaction id (TID), and the items, selected from a set I, purchased in this transaction. Each transaction contains a non-empty set of items from I, denoted as an *itemset* X_i. For example, the transaction t_2 contains the itemset $X_2 = \{Bread, Diaper, Beer, Eggs\}$.

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Table 1: A example of market-Basket transactions

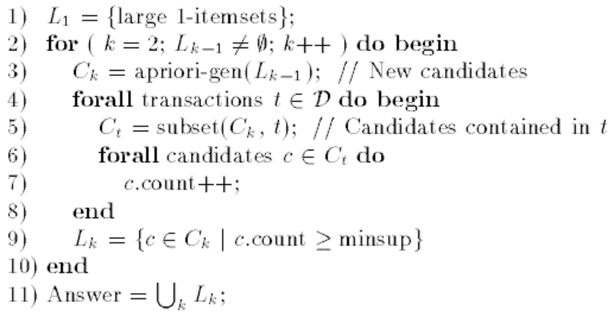
An itemset with k elements is called a *k-itemset*. $sup\_count(X)$ represents the number of transactions from database D where item X occurs:

$$sup\_count(X) = |\{t \in D | X \subseteq t\}|$$

A k-itemset X is said to be *frequent* if the support of X, $sup(X) = sup\_count(X)/|D|$, is greater than min_sup, a user-specified minimum support threshold. $|D|$ is the total number of transactions in D.

The first proposed method for itemset mining is the Apriori algorithm proposed by Agrawal and Srikant in 1994 [1]. Apriori proceeds level-wise. In the first step, the algorithm scans the database, searching for frequent items. Then it creates 2-itemset candidates, which are practically a combination of all possible pairs of frequent items (there are no duplicates and the order does not matter). From a second scan of the database, the algorithm decides which of these 2-itemsets are frequent or not, according to the min_sup threshold defined above. The candidate generation procedure continues with larger itemsets, incrementing by 1 the size of itemsets at each step, until it finally stops when there are no more frequent k-itemsets at some level k.

It is worth-noting that Apriori makes use of an important property, the so-called *anti-monotonicity* property of support with respect to itemset inclusion: all subsets of a frequent itemset are also frequent and, conversely, every superset of a non-frequent itemset is also non-frequent. The anti-monotonicity property is taken into account in the Apriori strategy in order to reduce the large number of candidates that are needed to be checked, the algorithm prunes all the supersets of an itemset candidate that is not frequent, as these supersets are not frequent as well. Figure 1 presents a pseudo-code of Apriori algorithm proposed by Agrawal and Srikant in [1]. However, there are some drawbacks of the method that we should take into consideration. Since Apriori makes multiple scans of the database, the run time of the algorithm may increase with the number of transactions. Also, in situations with prolific frequent patterns, long patterns, or quite low minimum support thresholds, for an Apriori-like algorithm it is costly to handle a huge number of candidate sets. The development of the FP-growth algorithm gives a solution to the generation of a huge number of candidates, by proposing a method for mining itemsets without need for candidate generation.



```
$L_1$ = {large 1-itemsets};
for ( $k = 2$; $L_{k-1} \neq \emptyset$; $k$++ ) do begin
    $C_k$ = apriori-gen($L_{k-1}$); // New candidates
    forall transactions $t \in \mathcal{D}$ do begin
        $C_t$ = subset($C_k$, $t$); // Candidates contained in $t$
        forall candidates $c \in C_t$ do
            $c$.count++;
    end
    $L_k = \{c \in C_k \mid c.\text{count} \geq \text{minsup}\}$
end
Answer = $\bigcup_k L_k$;
```

Figure 1: Pseudo-code of Apriori Algorithm (extract from [1])

Since then, itemset mining analysis has become a mature field of research with lots of proposed methods, but the Apriori algorithm still remains as one of the most popular approaches in itemset rules mining and other data-mining tasks. The state-of-the-art approach for itemset mining is LCM [18]. As our work is more focused on sequential pattern mining, we do not detail this algorithm.

Association rules Let I be the set of all possible items. An association rule is represented as an implication expression of the form $X \Rightarrow Y$, where X and Y are two itemsets, with $X \subseteq I$, $Y \subseteq I$ and $X \cap Y \neq \emptyset$. The above expression means that when an itemset X occurs in a transaction, it is most likely that the itemset Y occurs too. Two metrics are used to evaluate the quality of an association rule $X \Rightarrow Y$. The support (*sup*) of an association rule is the ratio of transactions that contain both X and Y: $sup\_count(X => Y) = sup\_count(X \cup Y)$

and the confidence of an association rule measures how often items in Y appear in transactions that contain also X. More formally:

$$conf(X => Y) = \frac{sup\_count(X \cup Y)}{sup(X)}$$

The association rule mining task is: given a transaction database D, a minimum support (min_sup) and a minimum confidence (min_conf) find all the pairs of disjoint itemsets X, Y, where $sup(X => Y) \geq min\_sup$ and $conf(X => Y) \geq min\_conf$. This mining task is done in two successive steps:

1. extract all the frequent itemsets $\mathcal{X}$ of the database, where $X \in \mathcal{X} \Leftrightarrow sup(X) \geq min\_sup$

2. for all $Z \in \mathcal{X}$, extract all the pairs of disjoint itemsets X, Y such that $X \cup Y = Z$ such that $conf(X => Y) \geq min\_conf$ (as Z is frequent, then $sup\_count(X => Y) \geq min\_sup$)

2.2 Sequential pattern mining

Clearly, the techniques described above do not cope with temporal information. The only information that we have for each transaction is the number (ID) which characterizes the transaction and also the items that are contained in the current transaction. The authors of Apriori are among the first scientists to take into account the timestamps of the transactions [17] by implementing sequential pattern mining techniques that include time information in the discovered patterns.

Here are some more definitions about terms in sequential pattern mining (borrowed from [7]). A *sequence* is an ordered list of itemsets and is denoted by $\langle s_1 s_2 \dots s_n \rangle$, where s_j is an itemset. A sequence $s_a = \langle a_1 a_2 \dots a_n \rangle$ is contained in another sequence $s_b = \langle b_1 b_2 \dots b_m \rangle$ if there exists n integers $i_1 < i_2 < \dots < i_n$ such that $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, ..., a_n \subseteq b_{i_n}$. The support of a sequence S_i, denoted by $sup(S_i)$, is the ratio of sequences in the database D that contain S_i.

For some sequence S_i, if $sup(Si) \geq min\_sup$ (minimum support given by the user), we say that S_i is a *frequent sequence* or a *sequential pattern*. It is worth mentioning here that in sequential pattern mining, it may have the occurrence of an item more than one time in the sequence, while in association rules mining each item may appear at most one time. Furthermore, the order of the itemsets in a sequence has an importance (this importance will be clarified with the example that follows), but in the case of itemsets, the order of items has no importance. These two observations about sequential patterns increase the complexity of the sequential pattern mining task, in contrast with the association rules mining task, and make it more challenging.

The Table 2 presents a database D of customer transactions. Each transaction consists again of the customer-id and the items purchased in the transaction, and an additional field, a timestamp representing the transaction time.

User ID	Transaction time	Items ID
1	12.02.1993	30
1	20.02.1993	50
2	08.02.1993	10, 20
3	10.02.1993	40
3	21.02.1993	30, 50, 70
3	29.02.1993	20, 30

Table 2: A temporal database

Each element of D is a sequence consisted of itemsets and for each itemset we have a transaction time. All the transactions of a customer can together be viewed as a sequence, where each transaction corresponds to a set of items, and the list of transactions, ordered by increasing transaction-time, corresponds to a sequence. We call such a sequence a customer-sequence. So, we can see the sequences from the transaction database of Table 2 on Table 3.

User ID	Sequence
1	$\langle (30), (50) \rangle$
2	$\langle (10, 20) \rangle$
3	$\langle (40), (30, 50, 70), (20, 30) \rangle$

Table 3: A sequence database

Returning now to the importance that has the order of the itemsets in a sequence, let us suppose that the item (30) is referred to the action "The client send an e-mail to his Internet supplier" and item (50) is referred to the action "The client call his Internet supplier". In this case, the thing that the client first sent an e-mail to the company about a problem that he fronts with the connexion and after some days he decided to call the company, it has an major importance for the company and the way its clients communicate with it. Respectively, if the phone call had been realised before the e-mail, this sequence of events would mean something different for the company. In contrast with the association rules mining, in the example of the client's market baskets, there is no use of the order of the events to the rules mining. The only thing that we are taking into account is the occurrence of items in the same itemset.

A first class of algorithms developed to solve the sequential pattern mining task is based on the Apriori property and its main representative is the *GSP* (Generalized Sequential Patterns) algorithm, proposed by Agrawal and Srikant [17]. The database used in the GSP algorithm has an horizontal format (the one presented in the previous section), while new algorithms were proposed, introducing the use of vertical format databases, in order to improve their performance. Among the most popular algorithms of this category is algorithm *SPADE* [20]. Other algorithms such as PrefixSpan [15] are based on the important principle of equivalent classes, where the search space is decomposed into smaller pieces processed independently in memory. According to experiments presented in [20], SPADE performs two orders of magnitude faster than GSP.

The problem with the Apriori type algorithms is that the number of generated candidates is, in the worst case, exponential and although methods such as constraints were introduced to alleviate the problem, the problem remains when the datasets are large. Another inherent problem is the repeated scans of the dataset. The recognition of these problems led to the construction of *Pattern Growth* algorithms by Han and Pei. Algorithm FreeSpan [10] aims to integrate the mining of frequent sequences with that of frequent itemsets and the use of projected sequence databases (e.g. introduced by FP-Growth [11]) to confine the search and growth of the subsequence fragments. After this proposal, Pei et al. proposed PrefixSpan [15] which is built 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. According to a study presented in [15], the authors point out that, in most cases, PrefixSpan performs better than GSP and SPADE.

2.2.1 Mining sequential pattern under constraints

As the number of frequent sequential pattern may quickly explode. Adding constraints on the extracted pattern reduces their amount, and while these constraints are also anti-monotonic the problem is not significantly harder. The use of such constraints has been successfully used in various applications such as text mining [2].

Pei et al. [16] defined seven types of constraints on patterns and embeddings. Constraints 1, 2, 3 and 5 are called "pattern constraints", while constraints 4, 6 and 7 are called "embedding constraints".

- Constraint 1 – item constraint. An item constraint specifies what are the particular individual or groups of items that should or should not be present in the patterns.
- Constraint 2 – length constraint. A length constraint specifies a prerequisite on pattern length. The maximal length constraint is anti-monotonic while the minimal length is not anti-monotonic.

- Constraint 3 – super-pattern constraint. A super-pattern constraint enforces the extraction of patterns that contain one or more given sub-patterns.
- Constraint 4 – aggregate constraint. An aggregate constraint is a constraint on an aggregation of items in a pattern, where the aggregate function can be *sum*, *avg*, *max*, *min*, *standard deviation*, etc. As an integrity constraint, this rule means that it is not possible to have a total amount lower than 1000 for pattern.
- Constraint 5 – Regular expression. Such a constraint is satisfied if the pattern is an accepted regular expression as stated by the user. A regular expression can be encoded in ASP as its equivalent deterministic finite automata. Expressing such a constraint is mainly technical and is not detailed here.
- Constraint 6 – Duration constraints. The duration (or span) of some pattern is the difference between its last item timestamp and its first item timestamp. A duration constraint requires that the pattern duration should be longer or shorter than a given time period.
- Constraint 7 – Gap constraints. A gap constraint specifies the maximal/minimal number of positions (or timestamp difference) between two successive itemsets in an embedding. The maximal gap constraint is anti-monotonic while the minimal gap is not anti-monotonic.

2.3 Temporal pattern mining

It is true that sequential pattern mining gives also information about the temporal sequence of the events, but this consideration is still limited as it provides no information about the interval between the events in a sequence. Many temporal mining algorithms proposed in the research community considers the temporal dimension in different ways, and here we briefly present some of the most famous.

A first approach, as defined in [14], is based on a new term - an episode. An episode is a partially ordered collection of events occurring together close in time. Given a set E of event types, an event is a pair (A, t), where $A \in E$ is an event type and t is an integer, the (occurrence) time of the event. An *event sequence* s on E is a triple (s, T_s, T_e), where $s = \langle (A_1, t_1), (A_2, t_2)....(A_n, t_n) \rangle$ is an ordered sequence of events, $T_s = t_1$ is called the starting time and $T_e = t_n$ the ending time of the event sequence.

The analysis of sequences cope with finding all frequent episodes from a class of episodes. To be considered interesting, the events of an episode must occur *close enough* in time. The user defines how close is *close enough* by giving the width of the *time window* within which the episode must occur. In addition to the width of the window, the user specifies in how many windows an episode has to occur to be considered frequent. Formally, a window on an event sequence $s = (s, T_s, T_e)$ is an event sequence $W = (w, t_s, t_e)$, where $t_s < T_e$ and $t_e > T_s$, and w is the time sequence where each event is happening in the interval $[ts, te]$. The *width* of the window is denoted as $w = t_e - t_s$. Mannila et al. propose in [14] an algorithm named WinEpi, the main idea of which is to find frequent events that happen within time-windows with user-defined width, or put into simpler words, this algorithm extracts episodes from an event sequence. In the same paper, the authors propose another algorithm, named MinEpi, which improves WinEpi: its pattern grammar is richer than the one of WinEpi and it needs to scan the database fewer times than WinEpi. On the other side, MinEpi is more memory consuming than WinEpi. Both WinEpi and MinEpi use Apriori-like strategy for tackling the mining problem.

Chen, Chiang and Ko present two efficient algorithms for mining time-interval sequential patterns [4]. The first algorithm, named *I-Apriori*, is based on the conventional Apriori algorithm, while the second one, *I-PrefixSpan*, is based on the PrefixSpan algorithm.

In [19], Yen and Lee introduce a new algorithm, called *TGSP* (mining Time-Gap Sequential Patterns) to discover not only the sequential patterns but also the time interval between any two items in the pattern. The authors call this kind of patterns time-gap sequential patterns. Basically, the algorithm takes as input a database of sequences and gives as result sequential patterns with numeric temporal intervals between them. For example, the pattern $\langle A[5-6]B[9-10]C\rangle$ (example borrowed from [19]) would suggest that event A is followed by event B after 5 to 6 time units and event B is followed by event C after 9 to 10 time units.

All methods described above do not take into account the duration of events, but they only give timestamps to the events but no information about their starting and ending points. The *QTIPrefixSpan* algorithm, proposed by Guyet and Quiniou [9], it is based on the PrefixSpan strategy and solves this problem by considering the quantitative duration of events. They introduce the term *temporal sequence* which is an ordered set of *temporal items*, defined as $A = (A, [l, u])$, where A is an event ID and l, u are timestamps which represent the beginning and the end of the event respectively. The main task of the algorithm is to extract frequent sequential patterns with temporal intervals characterizing event duration and relative event position in time.

3 Negative sequential analysis

Negative Sequential Patterns (NSP) refer to frequent sequences with non-occurring and occurring items, while Positive Sequential Patterns (PSP) are those that contain only occurring items. Moreover, NSPs focus on negative relationship between itemsets, where absent items are also taken into account. Given the example of [13], suppose we have a positive sequential pattern $p_1 = \langle abcdf\rangle$ and a negative one $p_2 = \langle ab\neg cef\rangle$, where a, b, c, d, e, f are events. Pattern p_1 means that events a, b, c, d and f usually happen in a row. But pattern p_2 further specifies that if events a and b happen in a row, and event c does not happen after them, then event e could happen instead of d.

According to [3] and [8], NSP mining has seen a very limited progress in recent years, thus all existing methods are quite inefficient and too specific for mining NSP. In addition, there is not a unified definition about negative containment (whether a data sequence contains a negative sequence). Negative sequential patterns can be formed by any combinations of negative and positive itemsets and so the problem becomes more complicated than in positive pattern mining. The addition of the appropriate constraints to negative sequences may solve the problem to some degree, because it is a way to reduce the number of Negative Sequential Candidates (NSC). However, the constraints that are used for sequential pattern mining, for example those described in paragraph 2.2.1, could not be applied to negative patterns. We can see that none of them could be adapted in negative sequential patterns and this is one reason for which the task of mining negative sequential patterns becomes more challenging. After having a look at the survey presented in [13], we can conclude that each one of the algorithms that already exist and are proposed for mining NSPs give different definition about negative containment and also use different constraints. According to this and to my knowledge, there are few methods available for NSP mining. The next paragraph presents the approaches and the algorithms that provide a base in NSP mining and are used in several scientific papers for further research work.

3.1 Negative sequential pattern mining

Among the methods that already exist and are developed for the negative sequential pattern mining task, we can distinct two approaches that could be characterized as typical as they share similar constraints and definitions. The elements that differ from the one approach to the other are the definitions about negative containment and the main algorithm. Before proceeding to these two algorithms, PNSP [12] and e-NSP [3], let us first give some general definitions used either from the one of the two algorithms or from both of them.

General definitions (The following definitions are borrowed from [12] and [3]) Given a sequence s, an occurring item is called a positive item and a non-occurring item is called negative item. The length of the sequence, $length(s)$, is the total number of items in all elements in the sequence. The size of a sequence, $size(s)$, is the total number of elements (itemsets) in the sequence. The negative itemset of itemset $I = (i_1, i_2, \ldots, i_m)$, denoted by $\neg I = \neg(i_1, i_2, \ldots, i_m)$, is referred to the absence of I, where all the items in I are absent altogether. A negative sequence consists of at least one negative item, respecting the constraints that will be described later.

The positive partner of a negative element $\neg e$ is e, denoted as $p(\neg e)$, i.e., $p(\neg e) = e$. The positive partner of a positive element e is e itself. The positive partner of a negative sequence $ns = \langle s_1 \ldots s_k \rangle$, denoted as $p(ns)$, changes all negative elements in ns to their positive partners. For example, $p(\langle a\neg(cd)\neg e\rangle) = \langle a(cd)e\rangle$. Element id is the order number of an element in a sequence. Given a sequence $s = \langle s_1 s_2 \ldots s_m \rangle$, $id(s_i) = i$ is the element identifier of element s_i. Element-id set $EidS_s$ of s is the set that includes all elements and their ids in s, i.e., $EidS_s = \{(s_i, id(s_i)) | s_i \in s, (1 \leq i \leq m)\} = \{(s_1, 1), (s_2, 2), \ldots, (s_m, m)\}$. The set including all positive and negative element-ids of a sequence s is called the positive and negative element-id set of s, denoted as $EidS_s^+$, $EidS_s$ respectively. Given a sequence s, for any subset $EidS_s = (a_1, id_1), (a_2, id_2), ..., (a_p, id_p) (1 < p \leq m)$ of $EidS_s$, $a = \langle a_1 a_2 \ldots a_p \rangle$, if $\forall a_i, a_{i+1} \in a (1 \leq i < p)$, we have $id_i < id_{i+1}$, then a is called an order-preserving sequence of $EidS_s$, denoted as $a = OPS(EidS_s)$. Taking the Example 3 of [3]), given $s = \langle\neg(ab)c\neg d\rangle$, its $EidS_s = (\neg(ab), 1), (c, 2), (\neg d, 3)$, $EidS_s^+ = (c, 2)$, $EidS_s^- = (\neg(ab), 1), (\neg d, 3)$. We can obtain $OPS(EidS_s^+) = \langle c\rangle$. Also, if $EidS_s = (\neg(ab), 1), (c, 2)$, we can create a sequence $OPS(EidS_s) = \langle\neg(ab)c\rangle$.

Sequence $s_a = \langle a_1 \ldots a_m \rangle$ is called a sub-sequence of the negative sequence $s_b = \langle b_1 \ldots b_n \rangle$, if there exists $1 \leq i_1 < i_2 < \ldots < i_m \leq n$, $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, $\ldots$, $a_k \subseteq b_{i_m}$. Sequence s_b is called a super-sequence of the negative sequence s_a. Taking the Example 4 of [3], given $s_b = \langle\neg(ab)cd\rangle$ and $s_a = \langle\neg(ab)d\rangle$, $EidS_{s_b} = \{(\neg(ab), 1), (c, 2), (d, 3)\}$, $EidS_{s_b} = \{(\neg(ab), 1), (d, 3)\}$ is a subset of $EidS_{s_b,s_a} = OPS(EidS_{s_b})$. A maximum positive sub-sequence is a special subsequence which is composed of all positive elements. Let $ns = \langle s_1 s_2 \ldots s_m \rangle$ be an m-size and n-neg-size negative sequence ($mn > 0$), $Sub(EidS_n^+ s)$ is called the maximum positive sub-sequence of ns, denoted as $MPS(ns)$. The support of a negative sequence ns, denoted by $ns.sup$, is the percentage of data sequences in DB containing ns. As presented in Example 5 of [3], given a negative sequence $s = \langle\neg(ab)cd\rangle$, $EidS^+ = \{(c, 2), (d, 3)\}$, its maximum positive sub-sequence is $MPS(s) = \langle cd\rangle$. A negative sequence s is a negative sequential pattern (NSP) if its support is not less than a threshold given by the user, called min_sup.

Constraints Two constraints are commonly enforced in the methods that follow:

- *Frequency constraint*: Given a negative sequence ns, the positive partner of each element in ns must be frequent. This constraint is defined slightly in a different way in the e-NSP approach, as it will be given later.

- *Format constraint*: Continuous negative elements in a NSC are not allowed. For example, $\langle\neg(ab)c\neg d\rangle$ satisfies format constraint, but $\langle\neg(ab)\neg cd >$ does not.

PNSP algorithm PNSP (Positive and Negative Sequential Patterns mining) [12] is the first algorithm proposed for mining both positive and negative sequential patterns. PNSP extends GSP, described in section 2.2, to deal with the mining of negative sequential patterns. The method in PNSP consists of three phases:

1. Mining all the positive sequential patterns, by using the GSP algorithm All the frequent positive itemsets are discovered as size-1 frequent sequences.

2. Positive itemsets are then used to generate all negative itemsets which satisfy a new term called missing frequency, denoted *miss_freq*, that assists a user to manage the interested number of negative itemsets in the sequence. For PNSP, a negative sequential pattern ns must satisfy the condition $support(ns) \geq min\_sup$, but it also has to satisfy the condition $miss\_freq \geq support(e_i)$, for each item e_i of the negative itemset.

3. Mining the negative sequential patterns. k-size negative sequential candidates are generated by appending a positive or negative frequent itemset to $(k-1)$-size candidate sequential patterns. Then, the sequence database is re-scanned to compute the support of these candidates.

During the 2nd phase of the candidate generation, after generating all negative itemsets, some candidates are pruned early before counting in the 3rd step. More precisely, the candidates whose maximum positive subsequence is not frequent are deleted. Candidates having negative subsequences without enough n-cover counts in DB are deleted too.

Neg-GSP algorithm Before continuing with another typical algorithm developed for mining negative patterns, the e-NSP algorithm, let us make a short reference to an older algorithm developed by almost the same group of the e-NSPs authors. Zheng et al., in [21], propose a negative version of the GSP algorithm to find negative sequential patterns. After trying to apply traditional pattern mining algorithms based on the Apriori principle, they came up with two main problems: the first one is that the Apriori principle can not be applied to negative sequential patterns, but it can simply specify that a sequence cannot be frequent if any of its sub-sequence is not frequent. The second problem has to do with the efficiency and the effectiveness of finding frequent patterns due to the vast candidate space.

e-NSP algorithm The authors of [3] claim that all the approaches proposed for mining NSPs so far are not efficient due to their need for several re-scans of the database, in order to calculate the support of negative candidates (NSC). In their method, they propose a set theory-based NSP mining and a corresponding algorithm, e-NSP (efficient NSP), to efficiently identify NSP by involving only the identified PSP, without re-scanning the database.

Using the frequency and format constraints described before, they only focus on the negative sequences ns whose positive partners are frequent, i.e., $sup(p(ns)) \geq min_sup$ (frequency constraint). They also define a third constraint, different from the third one of the PNSP algorithm, as follows:

Negative element constraint: The smallest negative unit in a NSC is an element. If an element consists of more than one item, either all or none of the items are allowed to be negative. As an illustration of the last definition, the sequence $\langle\neg(ab)cd\rangle$ satisfies the negative element constraint, but $\langle(\neg ab)cd\rangle$ does not because, in element $(\neg ab)$, only $\neg a$ is negative.

The framework and working mechanism of the proposed set theory-based NSP mining framework is illustrated in Figure 2 (Figure borrowed by [3]). In order to define the negative containment of this approach, we need a definition first, as given in [3]).

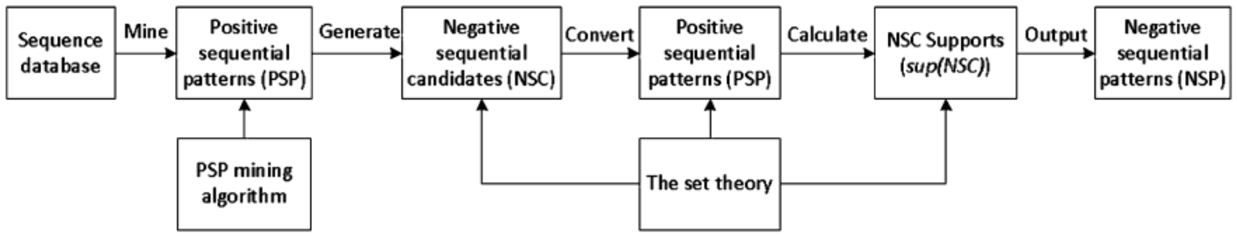

Sequence database
Mine
Positive sequential patterns (PSP)
Generate
Negative sequential candidates (NSC)
Convert
Positive sequential patterns (PSP)
Calculate
NSC Supports (sup(NSC))
Output
Negative sequential patterns (NSP)
PSP mining algorithm
The set theory

Figure 2: The set theory-based NSP mining framework

Definition 1 (First Sub-sequence Ending Position/Last Sub-sequence Beginning Position) *Given a data sequence $ds = \langle d_1 d_2 \dots d_t\rangle$ and a positive sequence α,*

1. *if $\exists p(1 < p \leq t), \alpha \subseteq \langle\langle d_1 \dots d_p\rangle \wedge \alpha \nsubseteq \langle d_1 \dots d_{p-1}\rangle$, then p is called the* First Sub-sequence Ending Position*, denoted as $FSE(\alpha, ds)$; if $a \subseteq \langle d_1\rangle$ then $FSE(\alpha, ds) = 1$;*
2. *if $\exists q(1 < q \leq t), \alpha \subseteq \langle d_q \dots d_t\rangle \wedge \alpha \nsubseteq \langle d_{q+1} \dots d_t\rangle$, then q is called the* Last Sub-sequence Beginning Position*, denoted as $LSB(\alpha, ds)$; if $a \subseteq \langle d_t\rangle$ then $FSE(\alpha, ds) = t$;*
3. *if $a \nsubseteq ds$, then $FSE(\alpha, ds) = 0$, $LSB(\alpha, ds) = 0$.*

Let us illustrate the last definition, by taking the example described in [3]: Given $ds = \langle a(bc)d(cde)\rangle$. $FSE(\langle a\rangle, ds) = 1$, $FSE(\langle c\rangle, ds) = 2$, $FSE(\langle cd\rangle, ds) = 3$, $LSB(\langle a\rangle, ds) = 1$, $LSB(\langle c\rangle, ds) = 4$, $LSB(\langle cd\rangle, ds) = 2$, $LSB(\langle(cd)\rangle, ds) = 4$.

Definition 2 (Negative Containment) *Let $ds = \langle d_1 d_2 \dots d_t\rangle$ be a sequence, $ns = \langle s_1 s_2 \dots s_m\rangle$ be an m-size and n-neg-size negative sequence, (1) if $m > 2t + 1$, then ds does not contain ns; (2) if $m = 1$ and $n = 1$, then ds contains ns when $p(ns) \nsubseteq ds$; (3) otherwise, ds contains ns if, $\forall(si, id(si)) \in EidS^{-}_{ns}(1 \leq i \leq m)$, one of the following three cases holds:*

1. *$(lsb = 1)$ or $(lsb > 1) \wedge p(s1) \nsubseteq \langle d_1 \dots d_{lsb-1}\rangle$, when $i = 1$;*
2. *$(fse = t)$ or $(0 < fse < t) \wedge p(sm) \nsubseteq \langle d_{fse+1} \dots d_t\rangle$, when $i = m$;*
3. *$(fse > 0 \wedge lsb = fse + 1)$ or $(fse > 0 \wedge lsb > fse + 1) \wedge p(si) \nsubseteq \langle d_{fse+1} \dots d_{lsb-1}\rangle$, when $1 < i < m$, where $fse = FSE(MPS(\langle s_1 s_2 \dots s_{i-1}\rangle), ds)$, $lsb = LSB(MPS(\langle s_{i+1} \dots s_m\rangle), ds)$.*

The example given in [3]) explains better the definition of negative containment. An important step of the e-NSP algorithm is the negative conversion strategy, which is based on the idea of converting the negative containment to positive containment, and then using the information of corresponding PSP to calculate the support of a NSC. For a negative sequence ns, its sub-sequences that include $MPS(ns)$ and one negative element e are called 1-neg-size maximum sub-sequences, denoted as $1 - negMS = OPS(EidS^{+}_{ns}, e)$, where $e \in EidS^{-}_{ns}$. The sub-sequence set including all 1-neg-size maximum sub-sequences of ns is called 1-neg-size maximum sub-sequence set, denoted as $1 - negMSS_{ns} = \{OPS(EidS^{+}_{ns}, e) | \forall e \in EidS^{-}_{ns}\}$. For example, if $ns = \langle \neg(ab)c\neg d \rangle$, then $1 - negMSS_{ns} = \{\langle \neg(ab)c \rangle, \langle c\neg d \rangle\}$.

The negative conversion strategy is described as follows. Given a data sequence $ds = \langle d_1 d_2 \dots d_t \rangle$ and an m-size and n-neg-size negative sequence $ns = \langle s_1 s_2 \dots s_m \rangle$, the negative containment problem can be converted to the following problem: the data sequence ds contains negative sequence ns if and only if the two conditions hold: (1) $MPS(ns) \subseteq ds$; and (2) $\forall 1 - negMS \in 1 - negMSS_{ns}, p(1 - negMS) \nsubseteq ds$. For example, let sequence $ds = \langle a(bc)d(cde) \rangle$. Firstly, if $ns = \langle a\neg dd\neg d \rangle$, then $1 - negMSS_{ns} = \{\langle a\neg dd \rangle, \langle ad\neg d > \}$, then ds does not contain ns because $p(\langle a\neg dd \rangle) = \langle add \rangle \subseteq ds$. Secondly, if $ns = \langle a\neg bb\neg a(cde) \rangle$, then $1 - negMSS_{ns} = \{\langle a\neg bb(cde) \rangle, \langle ab\neg a(cde) \rangle\}$. So, ds contains ns because $MPS(ns) = \langle ab(cde) \rangle \subseteq ds \wedge p(\langle abb(cde) \rangle \nsubseteq ds \wedge p(\langle aba(cde) \rangle) \nsubseteq ds$.

In this approach, the support of negative sequences is defined in a different way. Given an m-size and n-neg-size negative sequence ns, for $\forall 1 - negMS_i \in 1 - negMSS_{ns} (1 \leq i \leq n)$, the support of ns in sequence database D is:

$$sup(ns) = | ns | = | \{MPS(ns)\} \bigcup_{i=1}^{n} \{p(1 - negMS_i)\} |$$

For example, $sup(\langle \neg(ab)c\neg d \rangle) = sup(\langle c \rangle) - | \{\langle (ab)c \rangle\} \cup \{\langle cd \rangle\} |$.

If ns only contains a negative element, the support of ns is:

$$sup(ns) = sup(MPS(ns)) - sup(p(ns))$$

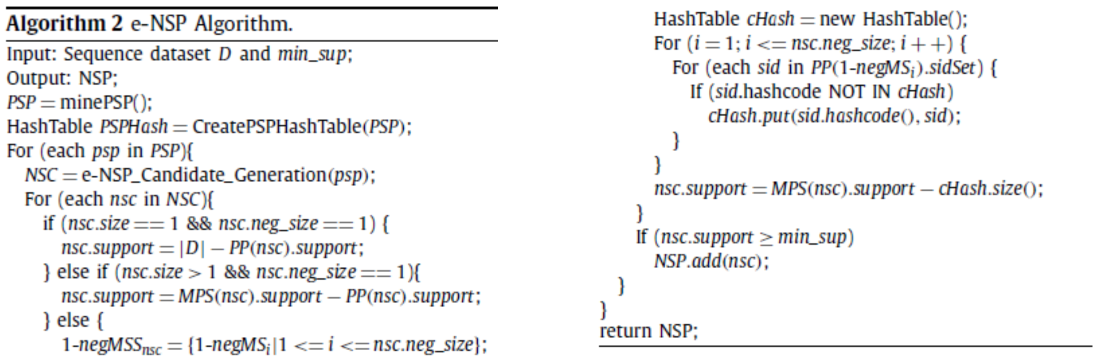
In particular, for the negative sequence $\langle \neg e \rangle$, $sup(\langle \neg e \rangle) = | D | - sup(\langle e \rangle)$

In [3], the Negative Sequential Candidate (NSC) Generation is defined as follows: for a k-size PSP, its NSC are generated by changing any m non-contiguous elements to their negative elements, $m = 1, 2, \dots, \ulcorner k/2 \urcorner$, where $\ulcorner k/2 \urcorner$ is a minimum integer that is not less than $k/2$. For example, the NSCs based on $\langle (ab)cd \rangle$ include, for $m = 1, \langle \neg(ab)cd \rangle, \langle (ab)\neg cd \rangle, \langle (ab)c\neg d \rangle$ and for $m = 2, \langle \neg(ab)c\neg d \rangle$.

Finally, in Figure 3 is presented a pseudo-code of the e-NSP algorithm proposed by Cao et al. in [3]. With minePSP function, all the Positive Sequential Patterns are first mined from the sequence database given at the entry, by using one of the PSP mining algorithms described in sections 2.3 and 2.3, such as GSP, PrefixSpan and SPADE.

4 Event prediction

Another task that has great importance in many applications is the prediction of future events. Such applications may be, for example, the prediction of customers purchases that can be a way to recommend items,the prediction of communication actions, like phone calls, of a client with a company that can be lead to the improvement of the communication between the client and the company, etc. The importance of such predictions is found either to the prediction of undesirable events or to the benefits that a frequent sequence of events may bring.



Algorithm 2 e-NSP Algorithm.

```
Input: Sequence dataset $D$ and $min\_sup$;
Output: NSP;
$PSP$ = minePSP();
HashTable $PSPHash$ = CreatePSPHashTable($PSP$);
For (each $psp$ in $PSP$){
  $NSC$ = e-NSP_Candidate_Generation($psp$);
  For (each $nsc$ in $NSC$){
    if ($nsc.size == 1$ && $nsc.neg\_size == 1$) {
      $nsc.support = |D| - PP(nsc).support$;
    } else if ($nsc.size > 1$ && $nsc.neg\_size == 1$){
      $nsc.support = MPS(nsc).support - PP(nsc).support$;
    } else {
      $1\text{-}negMSS_{nsc} = \{1\text{-}negMS_i | 1 <= i <= nsc.neg\_size\}$;
      HashTable $cHash$ = new HashTable();
      For ($i = 1$; $i <= nsc.neg\_size$; $i++$) {
        For (each $sid$ in $PP(1\text{-}negMS_i).sidSet$) {
          If ($sid$.hashcode NOT IN $cHash$)
            $cHash.put(sid.hashcode(), sid)$;
        }
      }
      $nsc.support = MPS(nsc).support - cHash.size()$;
    }
    If ($nsc.support \geq min\_sup$)
      $NSP.add(nsc)$;
  }
}
return NSP;
```

Figure 3: e-NSP Algorithm

Episode rules mining Similarly to association rules mining, which relies on the extraction of itemsets, *episode rules* can be extracted from episodes to predict events. We define as episode rule R the expression $R : P \Rightarrow Q$, where P and Q are two episodes, and which means that the episode Q appears after P. The *confidence* of the episode rule R expresses the probability to find Q after P and is defined as $conf(R) = supp(P \cup Q)/supp(P)$. The *support* $supp(P)$ of an episode represents the number of occurrences of P. R is to be said confident if its confidence exceeds a predefined threshold *minconf*. [6]

As mentioned by Fahed et al. in [5], the traditional methods for episode rules mining are not able to form rules where the consequent is temporally distant from the antecedent. As it is explained in [5], *these algorithms first form episodes from left to right by iteratively appending events temporally close to the episode being formed, in the limit of the predefined span. Then, the episode rule is built by considering the last element(s) of the episode as consequent of the rule. In addition, when forming these episodes, it is impossible to know if the event being appended will be part of the consequent or not, so it is impossible to constrain its distance to other events while forming the episode. The only way to mine rules with a consequent distant from the antecedent is to mine all rules and then filter the occurrences that do respect this distance. Due to the limited span, this distance cannot be large.* However, in most cases, the sooner an event is predicted, the more useful this prediction is for the person(s) concerned. In their work, Fahed et al. aim at predicting distant events, where the consequent of the episode rule is far from the antecedent, and also to predict events at an early stage, by accepting only small antecedents (in number of events and in time). The episode rules that have the characteristic of the small antecedent are called *minimal episode rules*.

Let us give here more details about the approach followed by Fahed et al. and the main steps of their algorithm, which is as efficient as the traditional ones are. The main difference from the traditional episode rules mining algorithms is the fact that here, it first determines the first event (prefix) of the rule, then it determines the consequent in a far distance from the prefix and finally, it completes the events of the antecedent (after the prefix). To succeed in the goals described before, the authors introduce a new concept for the window that includes a sequence and in which episode rules are searched. Let $Win(S, t_s, w)$ be a window in the sequence S of length w that starts at t_s, with its first element containing the prefix of an episode rule to be built (Definitions borrowed from [5]). Then, this window is splitted into three sub-windows as depicted in Figure 4. $Win(begin)$ is the segment of $Win(S, t_s, w)$ of length $w(begin) < w$, starting at t_s . $Win(begin)$

can be viewed as a way to limit the duration of the antecedent of an episode rule and guarantee that the antecedent occurs within a determined time. $Win(end)$ is a segment of $Win(S, t_s, w)$ of length $w(end) < w$, that ends at $ts + w$ and represents the time window of occurrence of the consequent. $Win(between)$ is the remaining sub-window of length $w(between)$, in which neither the antecedent nor the consequent can appear. $Win(between)$ guarantees the temporal distance between the antecedent and the consequent of an episode rule.

The algorithm of [5] consists of four steps. In the initialization phase, given an event sequence, the algorithm extracts all frequent events and their associated occurrence timestamps. After this, the episode rules are built iteratively by first fixing the prefix (the first event of the antecedent). Then, it searches for candidate consequents only in the $Win_(end)$ where the farthest candidates are located. And finally, the antecedent of the episode rule is completed, by adding iteratively all the frequent episodes on the right side of the prefix determined at the beginning of the episode rules construction.

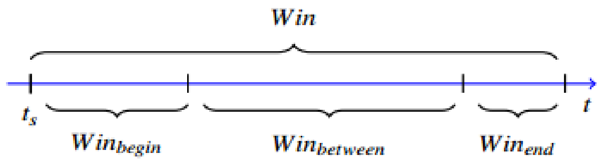


Figure 4: Sub-windows of Win(S,t,w)

Influencer events in episode rules The same team of researchers (Fahed et al.), who proposed the algorithm described in the previous paragraph, introduce, in [6], a new type of event, *influencer events*. This kind of event does not belong to the antecedent of an episode rule, but if it is introduced in the sequence it can disturb the occurrence of the consequent of the rule. For example, supposing we have an episode rule $R : ant \rightarrow p$. An event e is considered as an influencer event associated with the rule R, if some characteristics (such as the support or the confidence) of the new rule $R : ant \rightarrow e, p$ are significantly different from those of the rule R [6]. Influencer events present a high importance as they can be viewed as a way to impact the occurrence of events and according to the authors knowledge and to my knowledge, they have not been studied in the literature before. In addition, we can understand the relationship that exists between influencer events and negative patterns if we think about the role of a negative pattern. As it was presented in section 3, a negative pattern implies the non-occurence of one or more events. This means that if we are able to early predict an undesirable pattern, then we can try to find influencer events (negative/non-occuring events) that could be added in the previous undesirable pattern and so influence the behavior of the undesirable event. In other words, influencer events can help us prevent some events from happening.

According to the authors, there are three types of influencer events: *distance influencer events*, which influence the distance between the consequent and the antecedent of a rule, *confidence influencer events*, which influence the confidence of a rule and the *disappearance events*, that influence the support of a rule. To discover influencer events, they use basis rules with an antecedent as small as possible, related to the approach of the same authors described in the previous paragraph. The first part of their method relies on existing methods for extracting episode rules from episodes. Fahed et al. choose not to follow traditional algorithms for the episode rules mining task, because these algorithms require a high computational time and, also, they produce only frequent episode rules. So, rules with a low support are not taken into account and disappearance influencer events can not be discovered. They propose to use the algorithm described in [5], by taking advantage of two things: firstly, rules R are formed immediately after mining rules R, only by adding the influencer event e, and so the algorithm of discovering influencer events can be applied directly after the mining process; secondly, the new rule R may not be frequent and disappearance events can be discovered now.

5 Conclusion

In this report, we firstly presented the most important pattern mining methods, commonly used by the research field, that involve sequential patterns with or without temporal information. Then, we analysed negative sequential patterns, focusing on an algorithm used for efficiently extracting negative patterns and made an introduction to the notion of event prediction and to some methods that could be useful to our work.

As it derives from the existing pattern mining methods, none of them is quite sufficient for our task, because they do not take into consideration the inhibiting events that we would like to use in order to prevent the undesirable events. However, we are generally oriented towards an approach that combine positive/negative sequential patterns with patterns where the antecedent is early extracted and gives us the ability to early predict undesirable events.

The problematics of the internship are mainly summarized into three steps. We are first called to give a semantic definition of the negation for the timestamped event sequences and also give a definition for the inhibiting patterns. In addition, we will define early predictive negative patterns and introduce the concept of influencer event in negative predictive patterns. Then, we will have to design algorithms which will be able to extract such inhibiting patterns from a sequence database. Finally, evaluate these algorithms by using simulated or real data.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[2] Nicolas Béchet, Peggy Cellier, Thierry Charnois, and Bruno Crémilleux. Extraction de motifs séquentiels sous contraintes multiples (poster). In *13ème Journées Francophones Extraction et Gestion de Connaissances (EGC 2013)*, page 2 p., 2013.

[3] Longbing Cao, Xiangjun Dong, and Zhigang Zheng. e-nsp: Efficient negative sequential pattern mining. *Artificial Intelligence*, 235:156–182, 2016.

[4] Yen-Liang Chen, Mei-Ching Chiang, and Ming-Tat Ko. Discovering time-interval sequential patterns in sequence databases. *Expert Syst. Appl.*, 25(3):343–354, 2003.

[5] Lina Fahed, Armelle Brun, and Anne Boyer. Episode Rules Mining Algorithm for Distant Event Prediction. Research report, August 2014.

[6] Lina Fahed, Armelle Brun, and Anne Boyer. Influencer events in episode rules: A way to impact the occurrence of events. *Procedia Computer Science*, 60:527 – 536, 2015. Knowledge-Based and Intelligent Information, Engineering Systems 19th Annual Conference, KES-2015, Singapore, September 2015 Proceedings.

[7] Kiril Gashteovski. Temporal pattern mining and visualization of temporal patterns. Master's thesis, University of Rennes 1, 2014.

[8] Yongshun Gong, Chuanlu Liu, and Xiangjun Dong. Research on typical algorithms in negative sequential pattern mining. *Open Automation and Control Systems Journal*, 7:934–941, 2015.

[9] Thomas Guyet and René Quiniou. Extracting temporal patterns from interval-based sequences. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, IJCAI'11, pages 1306–1311. AAAI Press, 2011.

[10] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 355–359. ACM, 2000.

[11] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8(1):53–87, 2004.

[12] Sue-Chen Hsueh, Ming-Yen Lin, and Chien-Liang Chen. Mining negative sequential patterns for e-commerce recommendations. In *Asia-Pacific Services Computing Conference, 2008. APSCC'08. IEEE*, pages 1213–1218. IEEE, 2008.

[13] Sujatha Kamepalli, Raja Sekhara, and Rao Kurra. Frequent negative sequential patterns a survey. *International Journal of Computer Engineering and Technology (IJCET)*, 5, 3:115–121, 2014.

[14] Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data mining and knowledge discovery*, 1(3):259–289, 1997.

[15] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on knowledge and data engineering*, 16(11):1424–1440, 2004.

[16] Jian Pei, Jiawei Han, and Wei Wang. Constraint-based sequential pattern mining: The pattern-growth methods. *Journal of Intelligent Information Systems*, 28(2):133–160, 2007.

[17] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *International Conference on Extending Database Technology*, pages 1–17. Springer, 1996.

[18] Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. Lcm: An efficient algorithm for enumerating frequent closed item sets. In *Proceedings of FIMI*, 2003.

[19] Show-Jane Yen and Yue-Shi Lee. Mining non-redundant time-gap sequential patterns. *Applied intelligence*, 39(4):727–738, 2013.

[20] Mohammed J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, January 2001.

[21] Zhigang Zheng, Yanchang Zhao, Ziye Zuo, and Longbing Cao. Negative-gsp: An efficient method for mining negative sequential patterns. In *Proceedings of the Eighth Australasian Data Mining Conference-Volume 101*, pages 63–67. Australian Computer Society, Inc., 2009.