Faculty of Engineering and Information Technology
University of Technology, Sydney

Negative Sequential Pattern Mining

A thesis submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

by

Zhigang Zheng

January 2012

CERTIFICATE OF AUTHORSHIP/ORIGINALITY

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Candidate

_____

Acknowledgments

Foremost, I would like to express the deepest appreciation to my supervisor, Professor Longbing Cao, for his professional guidance, persistent help and continuous support throughout my Ph.D study and research.

I would like to thank Dr. Yanchang Zhao and Professor Xiangjun Dong, for their patient guidance, and scientific advice. Without their generous support this dissertation would not have been possible.

Besides, I offer my regards and blessings to all of my co-workers at lab, and thank them for their support in my research and during the completion of this dissertation.

In addition, I would like to thank all colleagues, specially Uma Srinivasan and Sue Bird, of the CMCRC HIBIS project, for their strong support for my research by providing much domain knowledge of health insurance industry.

Last but not the least, I would like to thank my family: my wife, my parents, and my son. Without their encouragement, finishing this dissertation would be impossible; without them, nothing would have any value.

Zhigang Zheng
June 2011 @ UTS

Contents

List of Figures

List of Tables

Abstract

Sequential pattern mining provides an important way to obtain special patterns from sequence data. It produces important insights on bioinformatics data, web-logs, customer transaction data, and so on.

Different from traditional positive sequential pattern (PSP) mining, negative sequential pattern (NSP) mining takes negative itemsets into account besides positive ones. It would be more interesting in applications where non-occurring itemsets need to be considered. This thesis reports our previous and the latest research outcomes in this area. The contributions of the thesis are as following.

- A comprehensive literature review of negative frequent pattern mining is described.
- A general framework of the NSP mining is proposed. It can be used to describe the big picture of both PSP and NSP mining problems.
- Three innovative algorithms are proposed to mine NSP efficiently.
- Extensive experiments about the three algorithms on either synthetic or real-world datasets show that the proposed methods can find NSP efficiently.
- A case study describes a real-life application on customer claims analysis in health insurance industry.

Three algorithms of NSP mining are proposed in this thesis, listed as below:

(1) The first algorithm *Neg-GSP* (Zheng, Zhao, Zuo & Cao 2009) is based on a PSP mining algorithm GSP (Srikant & Agrawal 1996). Neg-GSP deals with negative problem by introducing new methods of joining and generating candidates, which borrow ideas from GSP algorithm. And also, an effective pruning method to reduce the number of candidates is proposed as well.

(2) The second one is a Genetic Algorithm based algorithm (Zheng, Zhao, Zuo & Cao 2010), which is called *GA-NSP*. It is proposed to find NSP with novel crossover and mutation operations, which are efficient at passing good genes on to next generations. An effective dynamic fitness function and a pruning method are also provided to improve performance.

(3) The third algorithm *e-NSP* (Dong, Zheng, Cao, Zhao, Zhang, Li, Wei & Ou 2011) is based on the Set Theory. It mines NSP by only involving the identified PSP, without re-scanning the database. In this way, mining NSP does not require any additional database scans. It facilitates the existing PSP mining algorithms to mine NSP. It offers a new strategy for efficient mining of NSP.

The results of extensive experiments about the three algorithms show that they can find NSP efficiently. They have good performance compared with some other existing NSP mining algorithms, such as PNSP (Hsueh, Lin & Chen 2008).

If we compare the problem statements of the above three methods, *Neg-GSP* and *GA-NSP* share the same definitions, *e-NSP* uses stronger constraints since it requires clear boundary to follow the Set Theory. When comparing their performances, *GA-NSP* algorithm slightly outperforms *Neg-GSP* in terms of execution time, but it may miss some patterns in the complete result sets due to limitations of Genetic Algorithm. Apparently, *e-NSP* is the most efficient and effective one since it does not need to scan datasets to calculate the support of NSP. Although adding stronger constraints on e-NSP makes the search space much smaller than what it is under the normal definitions, it is still very practicable while being used in some real-life applications.

Following that, NSP mining case studies coming from health insurance industry are introduced. Based on real-life customer claims datasets, we use the proposed NSP mining methods to find PSP and NSP on solving two business issues, one is in ancillary service over-service analysis, another is fraud claim detection. Both of the two case studies demonstrate the benefits gained from mining NSP.

Chapter 1

Introduction

1.1 Background

Sequential pattern mining is an important task in data mining. It provides an effective way to get special patterns from sequence data. Finding sequential pattern has been widely recognized as a hot area in data mining and machine learning. It has been proven to be very useful or even essential while handling critical business problems, such as customer behavior analysis, event detection and bioinformatics. For example, it is widely employed in DNA, protein, and medicine identification, where it helps scientists to find out identical and different structures and functions of molecular or DNA sequences.

The concept of discovering sequential patterns was firstly introduced in 1995 (Agrawal & Srikant 1995), and aimed at identifying frequent subsequences as patterns in a sequence database, given a user-specified minimum support threshold. Some popular algorithms in sequential pattern mining include AprioriAll (Agrawal & Srikant 1995), Generalized Sequential Patterns (GSP) (Srikant & Agrawal 1996), PrefixSpan (Pei, Han, Mortazavi-Asl, Wang, Pinto, Chen, Dayal & Hsu 2004) and so on. GSP and AprioriAll are both Apriori-like methods based on breadth-first search, while PrefixSpan is based on pattern-growth strategy. Some other methods, such as SPADE (Sequential PAttern Discovery using Equivalence classes)(Zaki 2001) and SPAM

Table 1.1: A Transactional Data Table

Transaction Time	Customer ID	Buy Item
11-3-2009 11:00am	004	45
11-3-2009 11:00am	002	30,31,32
11-3-2009 12:00pm	004	29,16
11-3-2009 1:00pm	002	28
12-3-2009 7:00am	004	45
...	...	...
12-3-2009 7:01am	002	22,32

(Sequential PAttern Mining)(Ayres, Flannick, Gehrke & Yiu 2002), are also widely referred in the area of sequential pattern mining.

Sequence and Sequence Dataset

A sequence is an ordered list of elements like $< e_1\ e_2\ e_3\ \ldots\ e_n >$, where e_i is an element, and could be either one item or a set of items. The elements can be ordered by time, position or any other standard. Each element could also contain one or more items with no order between them. The length of a sequence is usually not fixed.

Sequence data is an important type of data which is popular in many scientific, medical, business service, bioinformatics, and some other applications.

An example of transactions data is shown in Table 1.1. In the data, customer 002, he/she has three transactions. If all of his/her transactions were ordered by the transaction time, they can be built into a sequence as $< (30, 31, 32)\ 28\ (22, 32) >$.

Another example comes from Bioinformatics. Following is a gene sequence which is ordered by position.

ACTGCTGCCAATC.

About Negative Sequential Pattern Mining

The previous literature work mainly focused on discovering the occurring items that form positive sequential patterns (PSP). However, it is increasingly recognized that negative sequential patterns (NSP), composed of both occurring and non-occurring items (Hsueh et al. 2008)(Lin, Chen & Hao 2007)(Zheng et al. 2009)(Ouyang & Huang 2007), can play an irreplaceable role in deeply understanding and tackling many business applications and problems.

In contrast to traditional PSP, NSP focus on negative relationships between itemsets, in which, non-occurring items are taken into consideration. More formal definitions of NSP mining problem are described in Chapter 3. Here we just give a simple example to illustrate the difference between PSP and NSP mining: suppose p_1=<a b c d> is a PSP; p_2=<a b $\neg c$ e> is a NSP; and each item, a, b, c, d and e, stands for a claim item code in the customer claim database of an health insurance company. By getting the pattern p_1, we can tell that an insurant usually claims for a, b, c and d in a row. However, with the pattern p_2 and only with it, we are able to find that given an insurant claim for items a and b, if he/she does NOT claim c, then he/she would claim item e instead of d. However, patterns like p_2 cannot be described or discovered by PSP mining.

Limited research on NSP mining can be identified in recent years (Ouyang & Huang 2007)(Lin et al. 2007)(Hsueh et al. 2008)(Zhao, Zhang, Cao, Zhang & Bohlscheid 2008) (Zhao, Zhang, Cao, Zhang & Bohlscheid 2009)(Zhao, Zhang, Wu, Pei, Cao, Zhang & Bohlscheid 2009)(Zheng et al. 2009)(Zheng et al. 2010).

1.2 Research Issues & Significance

While utilizing traditional frequent pattern mining algorithms for mining NSP, many problems stand in the way.

(1) Huge amounts of Negative Sequential Candidates (NSC) will be gen-

erated by classic breath-first search methods. Given 10 distinct positive items, If we only consider 3-item candidates, there are only 1,000 ($=10^3$) 3-item positive candidates, but there will be 8,000 ($=20^3$) 3-item negative candidates because there are another 10 negative items in addition to the 10 positive ones. It is a challenge to prune the large proportion of meaningless and unnecessary NSC. And it is important to develop efficient approaches for generating a limited number of genuinely meaningful NSC.

(2) Different negative supporting definitions challenge to NSP mining. There is a standard positive supporting definition about how a data sequence supports a positive sequence, but there is no unified negative supporting definition about how a data sequence supports or contains a negative sequence yet. For example, whether data sequence $<e>$ contains negative sequences $<e\ \neg e>$, $<\neg e\ e>$ or $<\neg e\ e\ \neg e>$, and whether $<a\ c>$ contains $<a\ \neg b\ c\ \neg d>$, different researchers give out different explanations (Hsueh et al. 2008)(Lin et al. 2007)(Zheng et al. 2009).

(3) Generally speaking, in spite of different negative supporting definitions, a data sequence can support much more negative candidates than positive ones. Take a 3-item data sequence $<a\ b\ c>$ as an example, it can only support 7 positive candidates $<a>$, $<b>$, $<c>$, $<a\ b>$, $<a\ c>$, $<b\ c>$ and $<a\ b\ c>$. But in the negative case, the data sequence $<a\ b\ c>$ can not only support the positive candidates as mentioned above, but also support a large bunch of negative candidates, such as $<a\ \neg a>$,$<b\ \neg a>$,$<b\ \neg b>$, $<a\ \neg a\ c>$, $<a\ \neg c\ c>$, $<a\ b\ \neg a>$, $<a\ b\ \neg b>$ and so on. As we know, a majority of those candidates will not appear even once on the sequence database. Even though we can prune meaningless and unnecessary NSC from them, there are still huge amounts of candidates after effective pruning.

(4) The Apriori principle doesn't apply to NSP mining. The Apriori principle can be simply described as: a sequence cannot be frequent if any of its sub-sequences is not. The Apriori principle is widely adopted to reduce the number of candidate subsequences in PSP mining (Agrawal & Srikant 1995)(Srikant & Agrawal 1996), but the principle does not work for patterns

containing negative items. Suppose c_1=<b $\neg c$> and c_2=<$b \neg c$ a> are two candidates, and s=<b d a c> is a data sequence. c_1 is a subsequence of c_2 since c_2 has one more element than c_1. We can see that s supports c_2 but doesn't support its subsequence c_1, which is to say, the candidate c_2 could have a greater support value than c_1 although c_1 is the subsequence of c_2, and so the Apriori principle cannot be adopted to reduce the growth of candidates in this case.

(5) In PSP mining, there are widely accepted definitions of positive sequences and positive patterns. However, this is not the case in NSP-related work. There is no such a wide recognition of what NSP is. Different definitions and constraints of NSP can be found in the literature (Hsueh et al. 2008)(Ouyang & Huang 2007)(Zhao et al. 2008)(Lin et al. 2007). A more consolidated formalization is certainly important for defining the underlying research issue.

(6) Furthermore, NSP mining has not become as manageable and workable as mining for PSP, due to intrinsic complexities. The existing NSP mining methods calculate the supports of NSC by additional scans of the database after identifying PSP, which leads to additional costs and results in low efficiency. It is essential to develop more efficient NSP mining methods without database re-scanning.

Hence, the above problems make NSP mining as an interesting research topic and an instructive exercise. Besides, NSP can play a more important role than PSP in some special applications, like the example of health insurance claim patterns in Section 1.1. Another real application comes from a previous research project of our lab about government welfare anomaly analysis (Zhao et al. 2008)(Zhao, Zhang, Cao, Zhang & Bohlscheid 2009)(Zhao, Zhang, Wu, Pei, Cao, Zhang & Bohlscheid 2009). From this project, we found that the concept of NSP is very suitable for some business issues. For example, failing to have a follow-up verification after a customer changes his address may more likely result in overpayment to the customer. Details and more application cases are also introduced in Chapter 7.

1.3 The Profile of Research Work

Figure 1.1 shows the research profile of the thesis.

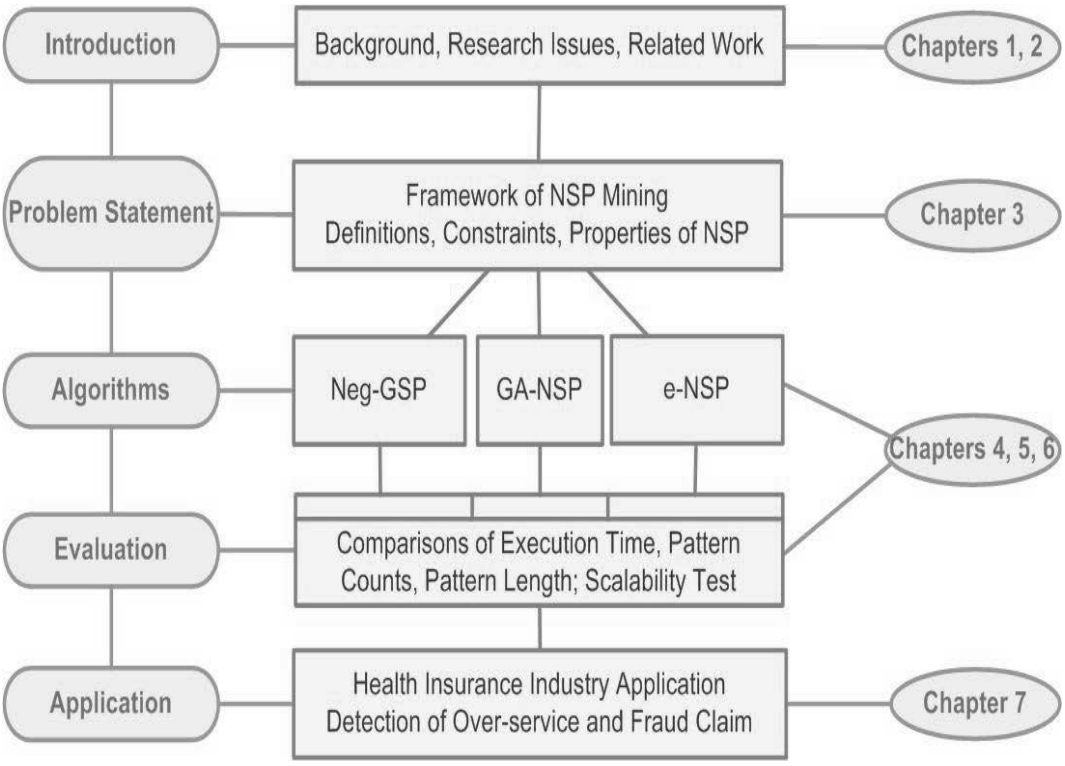


Figure 1.1: The Profile of Research Work

1.3.1 Research Introduction

First of all, to learn what kinds of research issues exist in NSP mining and how much effort has been made in this area are the most important step. After a comprehensive literature review, we found that only limited research on NSP mining can be identified in recent years. In the existing work, there are no general common definitions of NSP since it is hard to have a uniform formalization either.

1.3.2 Research Problem Statement

In PSP mining, there are a set of widely accepted definitions and clear problem statement about positive sequences and PSP mining. Therefore, we try to explore different definitions of NSP in existing research and give more general and formal definitions of NSP in Chapter 3. We bring out a general framework to describe NSP mining problem in Section 3.5 as well.

1.3.3 NSP Mining Methodologies

After giving formal problem statement of NSP, which represents both the framework and formal definitions, we propose three algorithms to mine interesting NSP. For each algorithm, detailed description and evaluation are provided. The evaluation focus on the comparison of their performance with other existing methods in terms of execution time test, pattern length and pattern counts test, and scalability test. Since there are few NSP methods available in existing research work, we can only find those which target at the research problems similar to our research topic. One of them is called PNSP (Hsueh et al. 2008). While evaluating the proposed methods, evaluation of the performance against PNSP is one of our tasks.

When we started this topic, we tried to establish some benchmark algorithms for our research on the topic. After reviewing most of existing classic PSP mining algorithms, and NSP mining algorithms. We choose PNSP (Hsueh et al. 2008), which targets on a similar research issue as ours. GSP algorithm is one of the most classic PSP mining algorithms, and therefore, we proposed a new algorithm, Neg-GSP, to mine NSP by extending GSP algorithm. Since Neg-GSP generates candidates by joining operation, which is more effective than the method used in PNSP, it can reduce the amount of invalid candidates to improve the performance.

Since Neg-GSP still generates huge amount of candidates and has to scan database to calculate support of each candidate, the performance can be improved further. Two steps could be improved to get better performance.

One is to find good NSP more efficiently, and another is to avoid scanning database when calculating support values of the candidates. Therefore, we then target on improving the two points.

Since evolution algorithms can find the optimal solutions/patterns by evolution, such as Genetic Algorithm, good/frequent sequences will be kept for next generation with higher priority. This feature would enable it to be much effective in mining either PSP or NSP. Based on this idea, a Genetic Algorithm based method, GA-NSP, was proposed. Experiments showed it outperformed Neg-GSP. GA-NSP is described in Chapter 5

At the same time, we tried to find more effective method to calculate support of candidates without scanning database. Going through a long way, e-NSP algorithm was proposed. It is based on the Set Theory and much efficiently in calculating support values of candidates. Chapter 6 has detailed information for it.

1.3.4 Experiments and Evaluation

For frequent pattern mining, the performance of an algorithm is always affected by the distribution of training dataset. Therefore, we generate synthetic sequence datasets as some of our test datasets by IBM data generator (Agrawal & Srikant 1995), which provides a function to generate various distribution datasets by predefined parameters composed of the following factors, see Table 1.2, and then use those datasets to test the performance of the proposed algorithms.

1.3.5 Case Studies

Real world applications are always the ultimate targets of data mining. We applied the NSP mining approach to many real world applications, including applications in the area of society welfare, health insurance and so on. Using the proposed NSP algorithms, we designed some solutions to deal with real-world business issues in an industry research project with a health insurance

Table 1.2: Synthetic Dataset Factors

C	Average number of elements per sequence
T	Average number of items per element
S	Average length of maximal potentially large sequences
I	Average size of items per element in maximal potentially large sequences
DB	Number of sequences (= size of Database)
N	Number of items

company in Australia, where the proposed methods are utilized to detect ancillary over-services and fraud claims.

1.4 Main Research Objectives

Proposing a general framework to formalize the NSP mining problem and creating new NSP mining methods are the main objectives in the thesis. Following are three aspects and ways that we tried to produce effective NSP mining methods.

- Improve PSP mining algorithms and adapt them for negative conditions;
- Deal with the NSP problem based on some classical data mining and machine learning methods, such as evolution algorithms, graph and so on;
- Find deducible theories to identify NSP based on the support information of its corresponding PSP.

1.4.1 Improving PSP Mining Algorithm

Some popular algorithms in PSP mining include AprioriAll (Agrawal & Srikant 1995), GSP (Srikant & Agrawal 1996), PrefixSpan (Pei et al. 2004),

SPADE (Zaki 2001), SPAM (Ayres et al. 2002) and so on. NSP mining extends the concept of PSP mining, making it somehow similar but different to PSP mining. Improving existing PSP mining algorithms and adapting them to be suitable for negative conditions is possible. However, while we apply positive mining algorithm to the negative cases, we need to improve them to deal with the following challenges:

(1) Negative candidates do not necessarily follow the Apriori principle. Some algorithms of generating candidates by breath-first search methods, such as GSP, are not suitable for generating negative candidates.

(2) Due to the vast candidate space, how can we find frequent patterns efficiently and effectively? The most popular method is to prune unnecessary candidates. But the pruning methods of PSP mining cannot deal with negative conditions if we borrow ideas directly from PSP mining.

Therefore, one objective of the thesis is to improve existing PSP algorithms and to adapt them for negative conditions.

1.4.2 Mining NSP Based on Genetic Algorithms

Besides improving PSP mining algorithms, proposing innovative algorithm for NSP mining produces much more challenges. The rough idea is to try some classical data mining and machine learning methods, such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), graph theory and so on, to find NSP.

The motivation is to mine NSP in a more effective search space, rather than the whole negative search space since the latter contains lots of unnecessary candidates.

GA is a good option to choose effective search space since it is efficient at passing good genes on to next generations and will focus on the space with good genes. But the issue comes along is that it is hard to find all the negative patterns because the GA-based method cannot guarantee the identification of all of them. Therefore, the goal of using GA to find NSP is to choose an effective search space and to find as many NSP as possible.

1.4.3 Mining NSP by Deducible Theories

The typical approach of getting support of NSC is to pass over the whole dataset which is obviously not efficient enough. We try to find a method to avoid additional passing over the dataset and therefore reduce execution time of calculating candidates' support values. A feasible method is to calculate supports of negative candidates by only using the support information of the corresponding PSP, without re-scanning the dataset.

To do that, we need to find the relationship between a negative candidate and its corresponding PSP, and to create precise deducible theories, or formulas, to calculate the support of a negative candidate based on the support of its corresponding positive patterns.

1.5 Research Contributions

The contributions of the thesis are:

- A comprehensive literature review of negative frequent pattern mining is described.
- A general framework of the NSP mining is proposed. It can be used to describe the big picture of both PSP and NSP mining problems.
- We propose a new algorithm, Neg-GSP, which is based on the GSP algorithm (Srikant & Agrawal 1996). The original joining and pruning steps of GSP are also improved to make them work for negative mining problem.
- A GA-based algorithm, GA-NSP, is proposed to find NSP efficiently. It obtains new generations by crossover and mutation operations, and uses dynamic fitness to control population evolution to find as many NSP as possible. A pruning method is also provided to improve performance.
- An innovative algorithm e-NSP, which can mine NSP by using only its corresponding PSP information, is proposed. In this way, there is no

need to re-scan the database after discovering PSP. Importantly, based on this strategy, most existing PSP algorithms can then be directly applied for NSP mining.

- Extensive experiments about the three algorithms on either synthetic or real-world datasets show that the proposed methods can find NSP efficiently and effectively.

- Real-world NSP mining applications are described to find over-service of ancillary services and detect fraud claims in health insurance industry.

1.6 Thesis Organization

This thesis is organized as follows.

Chapter 1 gives an introduction to NSP mining. It describes research issues, research framework, main research objects and our contributions on NSP mining.

Chapter 2 talks about the literature review of negative frequent pattern mining, including positive/negative association rules mining, PSP mining and NSP mining.

Chapter 3 presents formal and general descriptions of NSP definitions and a framework of NSP mining problem. Some criteria of negative supporting and the properties of NSP are defined.

Three innovative NSP mining algorithms, Neg-GSP, GA-NSP and e-NSP, are then given in Chapters 4, 5 and 6 correspondingly.

Chapter 7 introduces some real-world applications of NSP mining modeling in customer claims dataset of health insurance industry.

Chapter 8 shows the conclusion and future work.

Chapter 2

Related Work

The research on NSP mining follows those of PSP mining and negative association rules mining. And all of those NSP, PSP and negative association rules mining can be included in Frequent Pattern Mining. A frequent pattern is defined as a pattern, which can be a set of items, either with or without an order, occurs together in a database frequent enough to satisfy a certain minimum threshold (Han, Cheng, Xin & Yan 2007). Frequent pattern mining is the key step to find interesting patterns from databases, such as association rule mining, sequential patterns mining, etc, and is vital in data mining tasks.

Sequential pattern considers the order of itemsets, but association rule doesn't take that into account. For example, given a sequence, such as buying a *desktop* first, then an *laptop*, and then a *router*, if it occurs frequently in customers' shopping history with this special order, it is a (frequent) sequential pattern. When a frequent pattern only contains itemsets without any order, it becomes a classical association rule problem; for example, the same customer buys *desktop*, *laptop* and *router* without considering their orders.

2.1 Positive/Negative Association Rule Mining

2.1.1 (Positive) Association Rule Mining

(Positive) Association rule mining (Agrawal, Imieliński & Swami 1993)(Agrawal & Srikant 1994) is a commonly used method to generate rules in an unsupervised way. It finds interesting associations and/or correlation relationships among a large set of data items. The combinations of itemsets that occur frequently will be presented as association rules in the form of "if item/itemset A, then item/itemset B". A formal statement of the association rule problem is given below.

Let $I = \{I_1, I_2, \ldots, I_m\}$ be a set of m distinct attributes, also called *literals*. Let D be a data base, where each record T has a unique identifier, and contains a set of items such that $T \subseteq I$, an association rule is an implication of the form $X \rightarrow Y$, where X, Y $(X, Y \subset I)$ are sets of items called *itemsets*, and $X \cap Y = \Phi$. Here, X is called antecedent, and Y is called consequent (Agrawal et al. 1993).

In association rule mining, "support" and "confidence" are two measures to indicate the degree of uncertainty about a rule. Support is simply the number of transactions that include all items in the antecedent and consequent parts of the rule. Confidence is the ratio of the number of transactions that include all items in the consequent as well as the antecedent to the number of transactions that include all items in the antecedent as follows. The threshold values of support and confidence are usually used for filtering strong association rules.

$$Support(A \rightarrow B) = P(A \cup B) \tag{2.1}$$

$$Confidence(A \rightarrow B) = P(B \mid A) \tag{2.2}$$

For example, *Buys(x, desktop)* $\rightarrow$ *Buys(x, router) [support:70%, confidence:65%]* is an association rule, which means buying *router* with *desktop*

is a frequent pattern (if the predefined support threshold is less than 70%). Support=70% means there are 70% transactions that include both *desktop* and *router*. The confidence value of 65% means 65% of all transactions, which include *desktop*, also include *router*.

The association rule mining algorithms can be generally categorized into two types: Apriori-based and pattern-growth-based. In the next sections, we will discuss their concepts, advantages and disadvantages.

Breadth-First/Apriori-based Algorithms

Breath-first algorithms mainly consist of the Apriori algorithm and other Apriori-based methods. Apriori algorithm is based on the theory of the Apriori property. It was proposed by (Agrawal et al. 1993) to find frequent itemsets by generating candidate itemsets.

Apriori property: All nonempty subsets of a frequent itemset must also be frequent. And, for any infrequent itemset, its superset cannot be frequent. For example, $\{a, b, c\}$ is frequent, all of its sub-itemset like $\{a, c\}$ must also be frequent. If $\{b, c\}$ is infrequent, $\{a, b, c\}$ isn't frequent.

Based on this idea, the Apriori algorithm executes a breadth-first search through the data set of all itemsets. It starts from 1-item candidates at the very beginning, and then generates candidate itemsets C_{k+1} of size $k + 1$ by the patterns of size k iteratively. The Apriori procedure basically performs two phrases of actions: joining and pruning. In the joining phrase, patterns with k size, denoted by L_k, is joined with itself to generate potential candidates C_{k+1}. And then, in the pruning phase, all candidates which have one or more infrequent subsets should be pruned according to the Apriori property.

However, looking for candidates in each iteration is very time-consuming especially when the pattern set is huge. Some Apriori-base methods are proposed to improve its performance. (Agrawal & Srikant 1994) introduced a hash tree structure in 1994. The items in each transaction are sorted descendly in the hash tree. When it reaches one of its leaves, a candidate set, members of which share a common prefix, can be found easily. Then

only these candidates are joined for generating new candidates. Besides, (Sarracco 2003) summarized that partitioning (partitioning the data to find candidate itemsets) (Savasere, Omiecinski & Navathe 1995), transaction reduction (reducing the number of transactions to be scanned in future iterations), dynamic itemset counting (adding candidate itemsets at different points during a scan), and sampling (mining on a subset of the given data) also help to improve the performance of Apriori.

Breath-first-search-based algorithms typically have to scan the source database many times. Since it usually determines the support values of all (k-1)-item candidates before counting the support values of the k-item candidates. It must scan the source database once for each candidate.

Depth-First/Pattern-growth-based Algorithms

Another group of algorithms find frequent itemsets by the depth-first method, such as ECLAT (Zaki, Parthasarathy, Ogihara & Li 1997) and FP-Growth Algorithm (Han, Pei & Yin 2000).

FP-Growth uses a combination of the vertical and horizontal database layout to store the database in main memory. In a preprocessing step, FP-Growth derives a highly condensed tree, which is called FP-tree, to store the transactional data in main memory. Every item has a head list to link all transactions that contain that item. The generation of the FP-tree is done by counting occurrences and directly descending to some part of the itemsets in the search space. After the FP-tree is constructed, FP-Growth uses it to derive the support values of all frequent itemsets.

Both the Apriori and FP-Growth methods mine frequent patterns from a set of transactions in *TID-itemset* formats, where *TID* is a transaction-id and *itemset* is the set of items bought in the transaction with *TID*, known as horizontal data format.

ECLAT uses an *item-TID* set format, that is a vertical data format instead. ECLAT does not know all frequent itemsets at a level before starting the computation of the candidates at the next level. It combines depth-

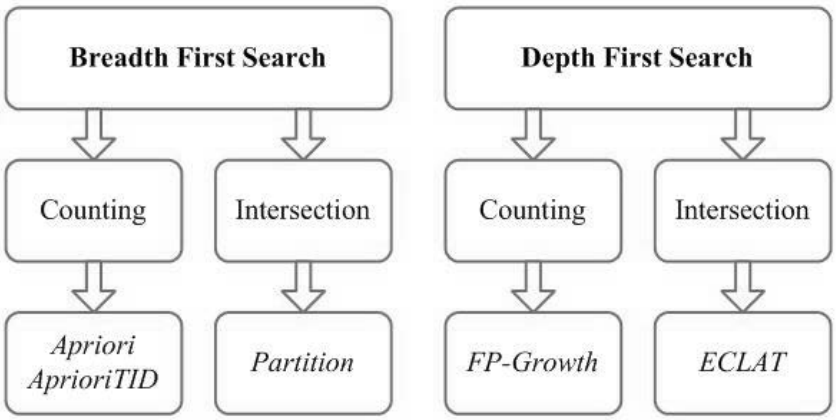


Figure 2.1: Systematization of Association Rule Mining

first search with *TID* list intersections, and recursively generates all itemsets with the same prefix using vertical database layout. ECLAT employs an optimization called "fast intersections". Whenever two *TID* Lists are intersected, the resulting *TID* List is interesting only if its cardinality reaches the threshold of minimum support *min_sup*. In other words, we should break off each intersection as soon as we are sure that it will not satisfy this threshold (Hipp, Güntzer & Nakhaeizadeh 2000).

Depth-first search will improve the memory utilization by maintaining fewer candidates and is more efficient than breath-first search while dealing with middle-size data set. Comparing with the depth-first algorithm, the breadth-first algorithm is less efficient but more scalable.

(Hipp et al. 2000) proposed a systematization of the above algorithms, see Figure 2.1.

2.1.2 Negative Association Rule Mining

After traditional association rule mining was proposed, some research work started to look at negative association rules mining. Earlier research in this area mostly focused on mining unexpected rules (Padmanabhan & Tuzhilin

1998) (Padmanabhan & Tuzhilin 2000) and exceptional rules (Suzuki & Shimura 1996) (Suzuki 1997) (Hussain, Liu, Suzuki & Lu 2000). They targeted at finding surprising patterns which show big difference from normal facts, which are also obtained from a model. For example, (Padmanabhan & Tuzhilin 1998) found an unexpected rule "in December, professionals shop more on weekdays than on weekends", which is different from a normal fact rule "professionals always shop on weekends".

In the literature, (Brin, Motwani & Silverstein 1997) was published in 1997 and is the earliest work in negative association rules mining to the best of our knowledge. It considered both absence and presence of items and used chi-square test to search correlation rules.

(Savasere, Omiecinski & Navathe 1998) proposed a method to find strong negative association rules by putting emphasis on domain knowledge, and defining strong negative association rules to limit search space to an accepted level and adding constraints. In a word, it tries to find the most interesting rules in a constrained search space. The constraints consider hierarchies of itemsets and their taxonomy. For example, *bottled water* (BW) and $frozen\ yogurt$ (FY) are two types of itemsets. Let BW_1 and BW_2 be two brands of bottled water, and FY_1 and FY_2 be two brands of frozen yogurt. Suppose the support of BW is 20000; FY is 30000, and support of BW & FY is 15000, BW_1 is 5000, BW_2 is 10000, FY_1 is 20000, FY_2 is 10000, then the *expected support* of BW_1 & FY_1 is as follows.

$$
\begin{aligned}
& esup(BW_1 \& FY_1) \\
&= sup(BW \& FY) \times \frac{sup(BW_1)}{sup(BW)} \times \frac{sup(FY_1)}{sup(FY)} \\
&= 15000 \times \frac{5000}{20000} \times \frac{20000}{30000} \\
&= 2500
\end{aligned}
$$

In the equation, $sup(BW \& FY)$ equals to 15000. That means the num-

ber of clients who bought both bottle water (BW) and frozen yogurt (FY) is 15000. Among those clients, expected probability of BW_1 is $sup(BW_1)/sup(BW)$, and expected probability of FY_1 is $sup(FY_1)/sup(FY)$. Having all those, the above equation can then get the expected probability of BW_1 & FY_1.

If $(esup(BW_1\&FY_1) - sup(BW_1\&FY_1))$ / $sup(BW_1) < min\_RI$, where min_RI is a predefined threshold, $BW_1 \rightarrow \neg FY_1$ is then an interesting negative rule.

(Wu, Zhang & Zhang 2004) and (Wu, Zhang & Zhang 2002) proposed a method for mining both positive and negative association rules. A rule follows one of the following formats: $<A \rightarrow B>$, $<A \rightarrow \neg B>$, $<\neg A \rightarrow B>$, $<\neg A \rightarrow \neg B>$. In the publications, they focused on infrequent itemsets as well as frequent ones. Frequent and infrequent itemsets of potential interest were defined. They argued that a rule $<X \rightarrow Y>$ is not interesting if $supp(X \cup Y) \approx supp(X) \times supp(Y)$. And it required that $interest(X, \neg Y)$ should be greater than a thereshold mi if $<X \rightarrow \neg Y>$ is an interesting negative association rules, which made the search space shrink significantly.

(Antonie & Zaïane 2004) also proposed an approach to mine both positive and negative association rules. It defined new interestingness different from the traditional *support-confidence* framework. And the approach can find confined negative association rules with strong negative correlation between X and Y in formats as $< \neg X \rightarrow Y >$, $< X \rightarrow \neg Y >$, $< \neg X \rightarrow \neg Y >$.

(Teng, Hsieh & Chen 2002) proposed a method named SRM (substitution rule mining) to mine negative association rules in the format of $< X \rightarrow \neg Y >$. It can be used to find what kinds of X can be a substitute for Y. A negative rule is defined as a substitution rule if $< X \rightarrow \neg Y >$ exists and X and Y have strong negative correlation. Here X and Y are *concrete* itemsets, which is verified by chi-square to see their dependency.

Mining negative association rules is a very challenging problem (Jean-Francois Boulicaut & Jeudy 2000). Boulicaut and Jeudy advised three possible approaches to deal with the negative problem. The first is a naive approach, which use both positive and negative itemsets directly and tra-

ditional association rules mining algorithms, like Apriori, to mine negative association rules. The second one is to derive negative association rules using only the information of positive rules. The third method is to use constraints to define interesting rules, and therefore reduce the search space, which is similar to the case that Apriori uses support as a constraint. The last one, which is to define interestingness, seems the most popular measures to deal with the problem. Interestingness is different from pure support-confidence measures, and many kinds of measures of interestingness have been proposed (Wu et al. 2004) (Antonie & Zaïane 2004) (Brin et al. 1997).

Besides the above three types of methods, some other methods are proposed, such as (Alata & Akin 2006) (Ouyang 2009). (Alata & Akin 2006) used Genetic Algorithm for mining negative quantitative association rules. It proposed a method using Genetic Algorithm to generate uniform initial population, and used an adaptive mutation probability and an adjusted fitness function, to mine negative quantitative association rule. (Ouyang 2009) used the fuzzy set theory to mine both positive and negative fuzzy association rules.

2.2 Sequential Pattern Mining

Frequent sequential pattern discovery can essentially be thought of as association rule discovery over a temporal database (Mabroukeh & Ezeife 2010). While association rule mining only considers frequent itemsets, without considering their orders, sequential pattern mining also discovers frequent ordered items/itemsets, such that the presence of a set of items is followed by another item in a time-ordered set of transactions.

Sequential pattern mining is an essential task in sequence data mining. It targets at finding frequent patterns among sequence database. The concept of discovering sequential patterns was first introduced in 1995 (Agrawal & Srikant 1995), and AprioriAll (Agrawal & Srikant 1995) algorithm was brought out at that work. After that, PrefixSpan (Pei et al. 2004) and GSP

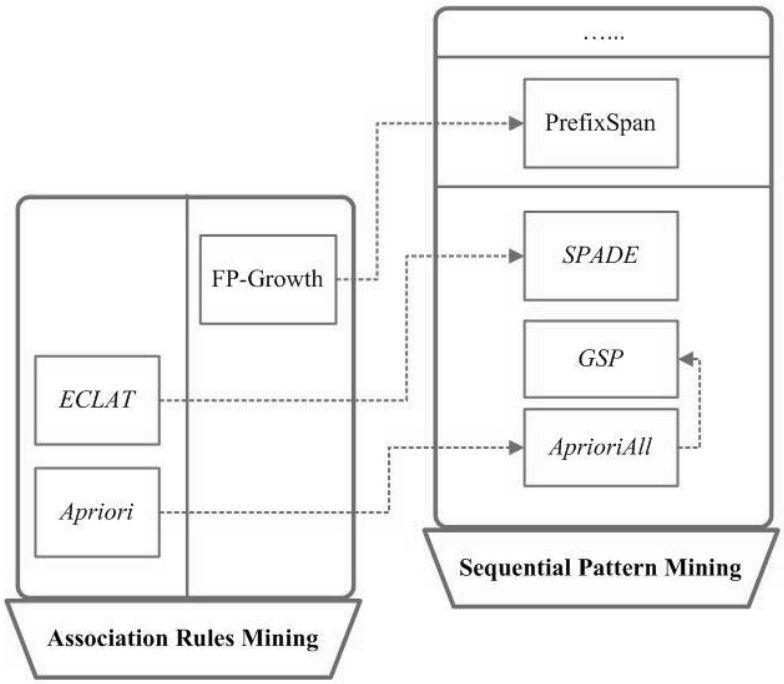


Figure 2.2: Association Rule And Sequential Pattern Mining Algorithms

(Srikant & Agrawal 1996) algorithms were proposed for sequential pattern mining. GSP and AprioriAll are Apriori-like methods that are based on breath-first search methods, while PrefixSpan is based on pattern growth. Other new methods such as SPAM (Ayres et al. 2002) are also popularly used at present.

2.2.1 Sequential Pattern Mining Algorithms

Many sequential pattern mining algorithms borrow and extend ideas from association rule algorithms, such as AprioriAll (Agrawal & Srikant 1995), GSP (Srikant & Agrawal 1996), PrefixSpan (Pei et al. 2004) and so on. Their relationships are demonstrated in Figure 2.2.

Apriori-Based Algorithms

AprioriAll (Agrawal & Srikant 1995) is a three-phase algorithm. Firstly, it finds all frequent itemsets, of which the support value is greater than a user-defined threshold - minimum support. In the second phase, it replaces each records in source database with the frequent itemsets contained by it. By doing that, it transforms source database to a new database. The last step is to identify sequential patterns in the new database. GSP (Generalized Sequential Patterns) (Srikant & Agrawal 1996) is a sequential pattern mining method that was developed by Srikant and Agrawal in 1996 and has been very popular since then. It is an extension of Apriori algorithm(Agrawal et al. 1993) for sequence mining. GSP uses a "Generating-Pruning" method and makes multiple passes over the data to find frequent sequences. GSP algorithm is very popular in sequential pattern mining. It makes multiple passes over the datasets. The first pass determines the support of each single item, and at the end of that pass, we will get all 1-item frequent sequences. Then those 1-item frequent sequences will be regarded as the seed set to generate candidate sequences for the next pass. Each candidate will have one more item than its seed sequence. And then, infrequent candidates are pruned without pass over the datasets. After getting new candidates, it passes over the data sets and get support of the new candidates. At the end of the pass, the algorithm identifies the frequent sequences from candidates. These frequent candidates become the seed for the next pass. The algorithm terminates when there are no more frequent sequences at the end of a pass, or when no candidate sequence is generated. Some other algorithms, such as AGM (Inokuchi, Washio & Motoda 2000), FSG (Kuramochi & Karypis 2001) and PSP (Masseglia, P. & Cicchetti 2000), also belongs to Apriori-based algorithms (Han et al. 2007)(Mabroukeh & Ezeife 2010).

Pattern-Growth Based Algorithms

The Apriori-like sequential pattern mining methods could generated a huge set of candidates in a large sequence database, and could scan databases

many times in training. It encounters difficulty when mining long sequential patterns. FreeSpan was proposed in (Han, Pei, Mortazavi-Asl, Chen, Dayal & Hsu 2000). Its general idea is to use frequent items to recursively project sequence databases into a set of smaller projected databases and grow subsequence fragments in each projected database. This process partitions both the data and the set of frequent patterns to be tested, and confines each test being conducted to the corresponding smaller projected database. Since FreeSpan reduce the expensive candidate generation and test substantially, it outperforms Apriori-based GSP algorithm considerably. PrefixSpan (Pei et al. 2004) is a depth-first algorithm that extends the pattern-growth approach for frequent pattern mining. It starts from frequent items with 1-item of the database, and then generates projected databases with their projection on the remaining data-sequences. The projected databases thus contain suffixes of the data-sequences from the original database, grouped by prefixes. The process is recursively repeated until no frequent item is found in the projected database.

Some other algorithms, such as gSpan (Yan & Han 2002), SPIN(Huan, Wang, Prins & Yang 2004), WAP-mine (Pei, Han, Mortazavi-asl & Zhu 2000), FS-Miner (El-Sayed, Ruiz & Rundensteiner 2004) and so on, are also mentioned as pattern-growth-based algorithms (Han et al. 2007)(Mabroukeh & Ezeife 2010).

Hybrid Algorithms

SPADE (Sequential PAttern Discovery using Equivalent classes) (Zaki 2001) is an Apriori-based sequential pattern mining algorithm that uses vertical data format. The main idea of SPADE is to cluster frequent sequences based on their common prefixes and enumerate the candidate sequences accordingly. By using a vertical data format with the creation of ID lists, SPADE reduces the number of scans of the sequence database. SPAM (Sequential PAttern Mining (Ayres et al. 2002)) algorithm combines the idea of the depth-first search, pattern-growth and pruning methods. The transactional data is

stored using a vertical bitmap representation, which allows for efficient support counting as well as significant bitmap compression. Using this kind of data structure, it achieves a high performance. A depth-first search strategy is used to generate candidate sequences, and various pruning mechanisms are implemented to reduce the search space. The integrated efficient pruning and indexing techniques enable the discovery of frequent patterns and are especially efficient when the sequential patterns in the database are very long. DISC-all (DIrect Sequence Comparison) (Chiu, Wu & Chen 2004) uses a new strategy to find frequent sequences without having to compute the support counts of non-frequent sequences. The main difference between the DISC strategy and Apriori-based methods is the way to prune non-frequent sequences. Apriori-based methods prune the non-frequent sequences according to the frequent sequences with shorter lengths. On the contrary, the DISC strategy prunes the non-frequent sequences according to the other sequences with the same length. It claimed that it can outperforms the PrefixSpan algorithm on mining in large databases (Chiu et al. 2004). LAPIN (LAst Position INduction) (Yang, Wang & Kitsuregawa 2007) is based on the ideas that the last position of an item is the key to judge whether or not a frequent k-length sequential pattern can be extended to be a frequent (k+1)-length pattern by appending the item to it. It can largely reduce the search space during the mining process, and is very effective in mining dense datasets. It claimed that it outperforms PrefixSpan on long pattern dense datasets.

2.2.2 Comparison & Computational Complexity Analysis

Comparing GSP with AprioriAll, the latter costs much more execution time in transforming the database and is not suitable for large data set since it needs to store the transformed database in memory. GSP algorithm generates fewer candidates than AprioriAll, so it is many times more efficient than AprioriAll. Although GSP could reduce the search space, it typically needs to pass over the database for multiple times, and suffers from generating a huge

set of candidate sequences, especially when mining long sequences. In another word, GSP spends lots of execution time in passing over database. The basic search methodology of SPADE and GSP is breadth first and Apriori pruning. Despite the pruning, both algorithms have to generate large sets of candidates. PrefixSpan is more efficient than GSP and AprioriAll.

It is not easy to determine the computational complexity of the algorithms since some of them have problems when the sequence database is large or the sequential patterns to be mined are numerous and/or long (Scime 2004). (Dong 2009) gave some explanation of computational complexity analysis for frequent sequential pattern mining. Theoretically, the problem of mining the complete set of sequential patterns is #P-complete [1]. Therefore, it is impossible to have a polynomial time algorithm unless P=NP. Even if P=NP, it is still unclear whether a polynomial time algorithm exists.

(Dong 2009) gave the computational complexity analysis of PrefixSpan as well. The major cost of PrefixSpan is the construction of projected databases. In the worst case, PrefixSpan constructs a projected database for every sequential pattern. If there do exist a good number of sequential patterns, the cost is non-trivial. Interestingly, we can show that the PrefixSpan algorithm is pseudo-polynomial. That is, the complexity of PrefixSpan is linear with respect to the number of sequential patterns, since each projection generates at least one sequential pattern, and the projection cost is upper bounded by the time of scanning the database once, and counting frequent items in the suffixes. Therefore, the computational complexity of PrefixSpan is $O(K)$ (K is the number of sequential patterns).

[1] #P-complete is a complexity class in computational complexity theory. A problem is #P-complete if and only if it is in #P, and every problem in #P can be reduced to it by a polynomial-time counting reduction, i.e. a polynomial-time Turing reduction relating the cardinalities of solution sets. (*http://en.wikipedia.org/wiki/Sharp-P-complete* 2012)

2.3 Negative Sequential Pattern Mining

2.3.1 PSP Mining Algorithms in NSP Mining Problem

However, the existing PSP mining algorithms, which were described in the above, are not suitable for NSP mining problem.

Firstly, down-closure property does not work in negative problems, which means that a sequence may still frequent if some of its subsequences are infrequent. More details and example are described in Section 3.4.3. Therefore, any of Apriori-based PSP mining algorithms can not deal with the negative mining problem.

Secondly, pattern-growth based methods does not work either. PrefixSpan is a good example, since the projected database of a node is not able to transfer all of the data sequences which support its child-nodes in negative pattern mining.

2.3.2 State-of-the-art Algorithms

Unlike PSP mining and negative association rule mining, there is very limited research work available in the literature on mining NSP. We will briefly introduce what we have identified.

The previous research work mainly focused on PSP mining. However, it is increasingly recognized that NSP, which are composed of both occurring and non-occurring items (Hsueh et al. 2008)(Lin et al. 2007)(Ouyang & Huang 2007)(Zheng et al. 2010)(Dong et al. 2011), can play an important role in deeply understanding and tackling many business problems. NSP focus on negative relationships between itemsets, in which, non-occurring items are taken into consideration. More formal definitions of NSP mining problem are described in Chapter 3.

Zhao et al. (Zhao et al. 2008) proposed an approach to mining event-oriented negative sequential rules from infrequent sequences. They derived equations to calculate supports, confidences and lifts of negative sequential

rules, and proposed an algorithm based on SPAM (Ayres et al. 2002). However the identified rules were limited to formats of $<A \rightarrow \neg B>$, $<\neg A \rightarrow B>$ and $<\neg A \rightarrow \neg B>$. They further presented an approach to discover both positive and negative impact-oriented sequential rules (Zhao, Zhang, Cao, Zhang & Bohlscheid 2009). Their work mainly mines negative sequential rules from frequent and infrequent positive patterns, and does not exactly involve NSP mining.

Ouyang & Huang (Ouyang & Huang 2007) extended traditional sequential pattern definition $<A\ B>$ to include negative elements such as $<\neg A\ B>$, $<A\ \neg B>$ and $<\neg A\ \neg B>$, which were similar to (Zhao et al. 2008). They put forward an algorithm which finds both frequent and infrequent sequences. It generates frequent itemsets first, then generates frequent and infrequent sequences, and finally obtains NSP from infrequent sequences. A drawback of the algorithm is that a large amount of space is required in order to find both frequent and infrequent sequences.

Nancy et al. (Lin et al. 2007) designed an algorithm PNSPM (Positive and Negative Sequential Pattern Mining) for mining NSP. They applied the Apriori principle to prune redundant candidates. They extracted meaningful negative sequences using the interestingness measure; nevertheless the above works defined some limited NSP, which are not general enough. According to their pattern definition, all elements must be positive except for the last one.

Sue-Chen et al.(Hsueh et al. 2008) proposed an algorithm called PNSP (Positive and Negative sequential pattern mining) for mining PSP and NSP in the form of $<(abc)\ \neg(de)(ijk)>$. They presented more comprehensive and more general definitions about NSP and extended GSP algorithm to deal with the NSP mining problem. Two concepts called *n-cover* and *n-contain* were employed to guide the method. It was claimed that if a *n-cover* value of a candidate was less than the predefined threshold *min_support*, any of its super-sequence was not going to be frequent and so the searching of candidate could be ended. They also proposed some constraints for NSP,

e.g., a valid negative sequence should not include contiguous non-occurring elements and data sequence s_d can't contain negative sequence s_n if the size of s_n is more than the size of s_d, and an itemset must be frequent positively to make a valid negative itemset.

PNSP is broken into three stages. 1) PSP are mined by traditional algorithms and all positive itemsets are derived from these PSP; 2) All negative itemsets are derived from these positive itemsets; 3). Finally, both positive and negative itemsets are joined to generate NSC, which are in turn joined iteratively to generate longer NSC in an Apriori-like way.

This approach calculated the support of NSC by additional scanning of the database again. When it generated NSC in the appending step, it produced a lot of unnecessary candidates. And also, we found out it may miss out some candidates.

Ouyang et al.(Ouyang, Huang & Luo 2008) presented the definitions of three types of negative fuzzy sequential patterns: $<a\ \neg b>$, $<\neg a\ b>$ and $<\neg a\ \neg b>$, and then described a method for mining native fuzzy sequential patterns from quantitative valued transactions.

2.4 Summary and Conclusion

Sequential pattern mining and association rule mining both belong to frequent pattern mining. The difference between them is that the former considers order but the latter doesn't consider it. NSP mining extends the research work of sequential pattern mining and negative association rule mining.

The association rule mining algorithms can be generally categorized into two categories: Apriori-based algorithms and pattern-growth based algorithms. Apriori(Agrawal et al. 1993) is an Apriori-based and classical breath-first search algorithm. FP-Growth(Han, Pei & Yin 2000) and ECLAT (Zaki et al. 1997) propose two popular pattern-growth based algorithms.

Negative association rule mining was firstly proposed in 1996 (Brin et al. 1997), and earlier researches in this area mostly focus on mining unexpected

rules (Padmanabhan & Tuzhilin 1998) (Padmanabhan & Tuzhilin 2000) and exceptional rules (Suzuki & Shimura 1996) (Suzuki 1997) (Hussain et al. 2000). Mining negative association rule is a very challenging problem (Jean-Francois Boulicaut & Jeudy 2000).

Sequential pattern mining is more complex than association rule mining since it considers orders of itemsets. Many classical algorithms, such as (Srikant & Agrawal 1996)(Pei et al. 2004)(Zaki 2001), have been proposed after sequential pattern mining was firstly introduced in 1995 (Agrawal & Srikant 1995). Existing sequential pattern mining algorithms can be categorized into three representative types, Apriori-based (Agrawal & Srikant 1995)(Srikant & Agrawal 1996)(Inokuchi et al. 2000)(Kuramochi & Karypis 2001)(Masseglia et al. 2000), pattern-growth based (Han, Pei, Mortazavi-Asl, Chen, Dayal & Hsu 2000)(Pei et al. 2004)(Yan & Han 2002)(Huan et al. 2004)(Pei et al. 2000)(El-Sayed et al. 2004) and hybrid methods (Zaki 2001)(Ayres et al. 2002)(Chiu et al. 2004)(Yang et al. 2007).

NSP mining considers non-occurring itemsets, which make the search space much bigger than traditional PSP mining. A few NSP mining methods (Ouyang & Huang 2007)(Lin et al. 2007)(Hsueh et al. 2008)(Zheng et al. 2009)(Dong et al. 2011) are available in the area.

The conclusions from the above literature review include:

- Pattern-growth based methods improve the memory utilization by maintaining fewer candidates and is more efficient than Apriori-based methods generally.

- NSP mining is a very challenge problem, with few literatures since it is a quite new topic. To the best of our knowledge, the NSP mining was firstly proposed in 2007 (Ouyang & Huang 2007)(Lin et al. 2007), and there are just few in literature talking about the problem so far, even if we include some papers which proposed for mining strong-constraint negative patterns.

- Boulicaut etc. (Jean-Francois Boulicaut & Jeudy 2000) advised three

possible approaches to deal with negative association rule mining problem. The first option uses both positive and negative itemsets directly and adapts traditional association rules mining algorithms to mining negative association rule. The second one is to derive negative association rules using only the information of positive rules. The third method is to use constraints to define interesting rules, and therefore reduce search space. These three possible approaches are also taken into consideration when we try to solve NSP mining problem in this thesis.

Chapter 3

Problem Statement

Traditional frequent sequential pattern mining has a common recognized problem statement, which is to find frequent sequential patterns on a sequence database by a given threshold. Accordingly, three essential concepts: sequence database, sequential patterns and supporting, have been defined very clear. Source transaction database is called *sequence database*, and each sequence in it is called a *data sequence*. A potential frequent pattern is called a *candidate* (*sequence*). If a candidate has higher support value than the threshold, it becomes a *pattern*. The *support* value of a candidate is counted according to how many data sequences in sequence database contain the candidate.

But it has much more complex descriptions in NSP mining problem.

Firstly, when negative items are taken into consideration, the amount of candidates will increase greatly. Therefore, all existing research work on NSP mining targets on interesting candidates only, which follow some different and special constraints.The constraints of negative candidates are very important to the topic.

Secondly, how to define supporting is another important issue, since different definitions of supporting may need different methodologies to solve them. When we verify supporting in positive problem, we only need to match items one by one between a data sequence and a candidate. But in negative

case, either matching positive item firstly, or negative item firstly, or just one by one by order, as what positive mining does, is not an easy job to describe and define formally. How to define supporting is a challenge in NSP mining problem.

Finally, PSP mining problem should be a sub-area of NSP mining problem, because NSP considers not only positive items, but also negative items. How to make PSP mining and NSP mining problems be compatible in a higher level?

Based on the above issues, some essential definitions should be given to make the problem clearer:

- Basic definitions of sequential pattern mining;
- The constraints of negative candidates;
- Criteria of supporting, which includes positive-first, negative-first and order-first supporting;
- Reconstructing data sequence and candidate sequence, to make NSP mining and PSP mining problem to be compatible in a general framework.

3.1 Basic Definitions

3.1.1 Positive/Negative Sequence

Definition 1: Item, Element and Sequence

Let I be a set of *items*, i.e. $\{x_1, x_2, \ldots, x_n\}$, which represents both positive items I^+ and negative items I^-, $I{=}I^+{\cup}I^-$. An *itemset* is a subset of I. A *sequence* s is an ordered list of itemsets, denoted by $<e_1\ e_2\ \ldots\ e_l>$, where e_j $(1{\leqslant}j{\leqslant}l)$ is an itemset. e_j is also called an *element* of the sequence, and denoted as $(i_1 i_2 \ldots i_m)$, where i_k is an *item*, $i_k{\in}I$ $(1{\leqslant}k{\leqslant}m)$. For simplicity, the brackets around an element are omitted if it only has one item, i.e., element

e_j=*(i)* is coded as e_j=*i*. An item is only allowed to occur at most once in one element, but can occur multiple times in several different elements of a sequence. Items in the same element are at same level, and the order is not significant among them. Usually we sort them by alphabetically. For example, $< (ab)\ c >$ equals to $< (ba)\ c >$ since a and b are in the same itemset and their order is not significant. Usually we just present it as $< (ab)\ c >$.

If an element e is composed of positive items only, it is called a *positive element*; if it is composed of negative items only, it is called a *negative element*; if both positive and negative items are included, the element is called a *mixed element*.

Note: 1) We always treat mixed elements as negative elements when there are not any special statements. 2) A negative element with more than one item is sometimes represented by a brief format, for example, a negative element *(¬a¬b¬c)* could be represented by *¬(abc)*.

If a sequence s is only composed of positive items, it is called a *positive sequence*. Otherwise, if it includes at least one negative item, it is called a *negative sequence*.

Example 1. Given a set of items I, where I=$I^+ \cup I^-$, I^+=$\{a,\ b,\ c\}$, and I^-=$\{\neg a,\ \neg b,\ \neg c\}$, we say that s_1=*<(ab) c a>* is a positive sequence and s_2=*<(ab) ¬c a>* is a negative sequence.

Definition 2: Length and Size of A Sequence

The *length* of sequence s, denoted as *length(s)*, is the total number of items in all elements in s. s is a *k-length* sequence or *k-item* sequence if *length(s)=k*.

The *size* of sequence s, denoted as $size(s)$, is the total number of elements in s. s is a *k-size* or *k-element* sequence if *size(s)=k*.

Example 2. Sequence s_2=*<(ab) ¬c a>* is composed of 3 elements *(ab)*, $\neg c$ and a, or 4 items a, b, $\neg c$ and a. s is a 4-length (4-item) and 3-size (3-element) sequence.

Definition 3: Subsequence

Sequence s_α=<α_1 α_2 ... α_n> is called a *subsequence* of sequence s_β=<β_1 β_2 ... β_m> and s_β is a *super sequence* of s_α, denoted as $s_\alpha \subseteq s_\beta$. If for each element α_i *(1*$\leqslant i \leqslant$*n)* in s_α, there exists j_i such that $\alpha_i \subseteq \beta_{j_i}$, and *1*$\leqslant j_1 < j_2 < \ldots < j_n \leqslant m$, also say that s_β contains s_α.

Example 3. <*a*>, <¬*c*>, <*b* ¬*c*> and <*(ab) a*> are all subsequences of <*(ab)* ¬*c a*>. <a ¬*b*> is a subsequence of <a (¬*b*¬*c*)>.

Definition 4: Maximum Positive Subsequence

Let s_α be a positive sequence and s_β be either a positive or a negative sequence, s_α is a *maximum positive subsequence* of s_β, if:

(1) s_α is a subsequence of s_β, and

(2) s_α includes all positive elements of s_β.

It is denoted by s_α=*MPS(*s_β*)*.

Example 4. <*a b f*> is the maximum positive subsequence of <*a b* ¬*c f*> and <*a b (*¬*c*¬*d) f*>. For a positive sequence, its maximum positive subsequence is itself.

Definition 5. 1-neg-size Maximum Subsequence

For a negative sequence s=<e_1 e_2 ... e_l>, its subsequence that includes $MPS(s)$ and only one more negative element e than *MPS(s)* is called a 1-*neg-size maximum subsequence*, denoted as *1-negMS*. It is a subsequence of s which is composed of all positive elements and one negative element. The subsequence set including all 1-neg-size maximum subsequences of s is called *1-neg-size maximum subsequence set*, denoted as *1-negMS*S_s.

Example 5.1. Given s=<*(*¬*a*¬*b) c* ¬*d*>, <*(*¬*a*¬*b) c*> and <*c* ¬*d*> are *1-negMS* of s, its 1-neg-size maximum subsequence set is *1-negMS*S_s= {<*(*¬*a*¬*b) c*>, <*c* ¬*d*>};

Example 5.2. Given *s'*=<¬*a (bc) d (*¬*c*¬*d*¬*e)*>, <¬*a (bc) d*> and <*(bc) d (*¬*c*¬*d*¬*e)*> are *1-negMS* of s', its 1-neg-size maximum subsequence

set is *1-negMSS*$_{s'}$={<¬*a* (*bc*) *d*>, <(*bc*) *d* (¬*c*¬*d*¬*e*)>}.

Definition 6. Reverse Partner

The reverse partner of a positive element e is $\neg e$, the reverse partner of a negative element $\neg e$ is e, denoted as $RP(e)=\neg e$ and $RP(\neg e)=e$.

The reverse partner of a sequence $s=<e_1 \ldots e_l>$ is to change all elements in s to their corresponding reverse partners, denoted as $RP(s)$, i.e., $RP(s)=\{<e'_1 \ldots e'_l> |\ e'_i=RP(e_i),\ e_i \in s\}$.

Example 6. *RP*(<(¬*a*¬*b*) *c* ¬*d*>)=<(*ab*) ¬*c d*>.

3.1.2 Data Sequence & Candidate Sequence

Definition 7: Data Sequence

A *sequence database D* is the source dataset for model training, denoted as a set of tuples [*sid*, s_d], where *sid* is a *sequence id* and s_d is a *data sequence*. Sequence database only contains positive data sequences, and normally there are not any negative data sequences in it.

Definition 8: Candidate Sequence

A sequence proposed to be a potential pattern is called *candidate sequence*. A candidate sequence could be either positive or negative.

3.2 Constraints of Positive/Negative Candidates

It is too costly and infeasible to talk NSP mining without any constraints, since the search space is simply too huge and most identified patterns based on free combinations are meaningless. In practice, constraints have to be considered, as shown in all available references (Ouyang & Huang 2007)(Hsueh et al. 2008)(Dong et al. 2011)(Lin et al. 2007).

There are many kinds of constraints for negative candidates in existing research works. For example, (Ouyang & Huang 2007) proposed NSP by the format of $<A\ \neg B>$, $<\neg A\ B>$ and $<\neg A\ \neg B>$, where A and B are positive frequent subsequence and $A \cap B = \varnothing$. (Lin et al. 2007) only focuses on NSP which has a negative item at the end of each pattern, like $<c_1\ c_2\ \ldots\ \neg c_m>$, where c_i $(1 \leqslant i \leqslant m)$ is a positive item.

Different constraints could make different definition of NSP, so as to different support calculating methods and different heuristic pruning measures. Some common constraints, which is popularly used in other research works, are listed as following.

Candidate Constraint 1: To Be Positive Frequent

For each negative item in a NSP, its corresponding positive item is required to be frequent. For example, *(¬i)* is an interesting negative item if its positive item *(i)* is frequent.

The reason is that we are always interested in non-occurrences of frequent positive items. It is helpful for us to avoid searching in the unnecessary search space. This constraint is the most popular and valuable one (Ouyang & Huang 2007)(Lin et al. 2007)(Hsueh et al. 2008)(Zheng et al. 2009).

Besides the above common constraint, we list some other constraints on interesting NSP from existing research work.

Candidate Constraint 2: Format Constraints

(1) *Two or more adjacent negative elements are not accepted in a negative sequence.*

Example 7. *<(¬a¬b) c ¬d>* satisfies the constraint, but *<(¬a¬b) ¬c d>* does not since *(¬a¬b)* and *¬c* are two adjacent negative elements.

This constraint is used by many researchers (Hsueh et al. 2008)(Zheng et al. 2009)(Dong et al. 2011).

(2) *Only special parts of elements in the sequence can be negative.*

(Ouyang & Huang 2007) only accepted a negative candidate with the last part of it being negative. For example, $<(ab)\ \neg c\ \neg d>$ was accepted. (Lin et al. 2007) had a more strict constraints and proposed that only the last element of a negative candidate can be negative, such as $<(ab)\ c\ \neg d>$.

3.3 Reconstruct Data Sequence

Based on the definition of *positive sequence*, we added negative items to the definition of *negative sequence*. A negative item x represents that its reverse partner $RP(x)$ did not occur, that is to say, it can cover all the other positive items except $RP(x)$. For example, following the Example 1, given the whole positive items set I^+=$\{a,\ b,\ c\}$ and a negative item $x = (\neg a)$. x means that positive item a can not occur, but the other positive items can occur. Therefore, given a negative item $x = (\neg a)$, it can also be described by $x' = (b\ or\ c)$.

On the other side, a positive item $x = (a)$ represents positive item a occur, that is to say, the other items did not occur. For example, given the whole positive items set I^+=$\{a,\ b,\ c\}$ and positive item $x = (a)$. x means that positive item a occurred, but the other two positive items, b or c did not occur. Therefore, given a positive item $x = (a)$, it can also be described by $x' = (\neg b\ and\ \neg c)$.

In the above two examples, x' is called the equivalent item of x.

Definition 9: Equivalent Element

If an element e has the same meaning and coverage as another element e', e and e' are called *equivalent elements*. We say e is an equivalent element of e', or e' is an equivalent element of e, denoted by $e \equiv e'$, or $e'=EE(e)$. If e is not an equivalent element of e', it is denoted by $e \not\equiv e'$.

Example 8. Given a set of items I=$\{a,\ b,\ c,\ d,\ \neg a,\ \neg b,\ \neg c,\ \neg d\}$ and an element $e=(ab)$, its equivalent elements could be (ab), $(ab\neg c)$, $(ab\neg d)$, $(\neg c\neg d)$ and $(ab\neg c\neg d)$.

Table 3.1: Example of Maximum Equivalent Sequence

Sequence	Element1	Element2	Element3
s=<*(ab) c a*>	*(ab)*	*c*	*a*
MES(s)=<*(ab¬c) (¬a¬bc) (a¬b¬c)*>	*(ab¬c)*	*(¬a¬bc)*	*(a¬b¬c)*

Definition 10: Maximum Equivalent Element

Given two elements e and e', $e \equiv e$', if there are not any other equivalent elements of e that have more items than e', we call e' the *maximum equivalent element* of e, denoted by *e'=MEE(e)*.

Example 9. Following Example 8, *(ab¬c¬d)* is the maximum equivalent element of e.

Definition 11: Maximum Equivalent Sequence

Given two sequences s=<e_1 e_2 ... e_l> and s'=<e'_1 e'_2 ... e'_l>, if $\forall m$ *(0<m⩽l)*, e'_m=*MEE*(e_m), s' is the *maximum equivalent sequence* of s, denoted by *s'= MES(s)*.

Example 10. Given a set of items I, where I=$I^+ \cup I^-$, I^+={*a, b, c*}, and I^-={*¬a, ¬b, ¬c*}, a sequence s=<*(ab) c a*>, according to the above definition, we get *s'=(ab¬c) (¬a¬bc) (a¬b¬c)*> and *s'=MES(s)*. Table 3.1 presents a more straightforward view.

According to the definition of maximum equivalent sequence, a data sequence is always a subsequence of its maximum equivalent sequence. It is also obvious that a data sequence only has parts of subsequences of what its maximum equivalent sequence has.

To describe the NSP mining problem, we extend sequence database to a new database which is composed of its corresponding maximum equivalent sequences, that is to say, for each tuple [*sid*,s_d], it is extended to [*sid, MES*(s_d)].

Table 3.2: Examples of Sequence Containing

Data Sequence s_d	Maximum Equivalent Sequence s'_d	Basic Operation	Candidate Sequence s_c
$<(ab)\ c\ a>$	$<(ab\neg c)\ (\neg a\neg bc)\ (a\neg b\neg c)>$	$\sqsubset$	$<a\ \ c\ \ \neg b>$, $<(b\neg c)\ \neg a>$, $<\neg b\ a>$
$<(ab)\ c\ a>$	$<(ab\neg c)\ (\neg a\neg bc)\ (a\neg b\neg c)>$	$\not\sqsubset$	$<\neg c\ \ b>$, $<b\ c>$, $<b\ \ c\ (\neg a\neg c)>$

Table 3.3: Examples of Sequence Absolutely Containing

Data Sequence s_d	Maximum Equivalent Sequence s'_d	Basic Operation	Candidate Sequence s_c
$<(ab)\ c\ a>$	$<(ab\neg c)\ (\neg a\neg bc)\ (a\neg b\neg c)>$	$\sqsubseteq$	$<a\ c\ \neg b>$
$<(ab)\ c\ a>$	$<(ab\neg c)\ (\neg a\neg bc)\ (a\neg b\neg c)>$	$\not\sqsubseteq$	$<(b\neg c)\ \neg a>$, $<\neg b\ a>$

3.4 Supporting

3.4.1 Basic Operation

Basic Operation 1: Contain

A data sequence s_d=$<d_1\ d_2\ \ldots\ d_l>$ *contains* a candidate sequence s_c=$<c_1\ c_2\ \ldots\ c_m>$, denoted by $s_c \sqsubset s_d$, if s_c is a subsequence of *MES(s_d)*. That is to say, $\exists$ sequence s'_d=$<d'_1\ d'_2\ \ldots\ d'_l>$, where s'_d=*MES(s_d)*, and for each element c_i *($1 \leqslant i \leqslant m$)* in candidate sequence s_c, there exists an integer j_i such that: $c_i \subseteq d'_{j_i}$, and *($1 \leqslant j_1 \leqslant j_2 \leqslant \ldots \leqslant j_m \leqslant l$)*.

Example 12. Given a set of items I, where I=$I^+ \cup I^-$, I^+=$\{a,\ b,\ c\}$ and I^-=$\{\neg a,\ \neg b,\ \neg c\}$. Table 3.2 shows some examples of sequence containing.

Based on the above definition, we define an extended definition of absolutely containing.

Basic Operation 2: Absolutely Contain

A data sequence s_d=<d_1 d_2 ... d_l> *absolutely contains* a candidate sequence s_c=<c_1 c_2 ... c_m>, denoted by $s_c \sqsubseteq s_d$, if:

$\exists$ sequence s'_d=<d'_1 d'_2 ... d'_l>, where s'_d is the maximum equivalent sequence of s_d. s'_d contain s_c and s'_d doesn't contain *RP(s_c)*;

Example 13. Following Example 12, some examples of absolutely containing are shown in Table 3.3. <*(ab) c a*> doesn't absolutely contain <*(b¬c) a*> or <*¬b a*> because it contains their reverse partners <*(¬bc) ¬a*> or <*b ¬a*>.

3.4.2 Criteria of Supporting

Using the above two basic operations, we define three criteria of supporting as follows:

Criteria 1: Positive-First Supporting

Under this criteria, positive elements have higher priority than that of negative ones when verifying whether a data sequence supports a candidate sequence.

Given a data sequence s_d=<d_1 d_2 ... d_m> and a candidate sequence s_c=<c_1 c_2 ... c_n>, sequence s'_d=<d'_1 d'_2 ... d'_m>=*MES(s_d)*, we call s_d *supports* s_c if:

- For each positive supper element e_j *(1⩽j⩽l)* in s'_c, there exists integers k_{j_1} and k_{j_2} such that: <e_j>⊏<$d'_{k_{j_1}}$, ..., $d'_{k_{j_2}}$>,

  and (1) *(1⩽k_{i_1}⩽k_{i_2}<k_{j_1}⩽k_{j_2}⩽l)* if e_j has a positive super element e_i right before it.

  and (2) *(1⩽k_{j_1}⩽k_{j_2}<k_{i_1}⩽k_{i_2}⩽l)* if e_j has a positive super element e_i right after it.

- For each negative supper element e_j *(1⩽j⩽l)* in s'_c,

  (1) if *1<j<l*, ∃ integers $k_{(j-1)_2}$, $k_{(j+1)_1}$, $e_j \sqsubseteq < d'_{k_{(j-1)_2+1}}, \ldots, d'_{k_{(j+1)_1-1}} >$.

  (2) if *j=1*, ∃ an integer $k_{(j+1)_1}$, $e_j \sqsubseteq < d'_1, \ldots, d'_{k_{(j+1)_1-1}} >$.

  (3) if *j=l>1*, ∃ an integer $k_{(j-1)_2}$, $e_j \sqsubseteq < d'_{k_{(j-1)_2+1}}, \ldots, d'_l >$.

  (4) if *j=l=1*, $e_i \sqsubseteq < d'_1, \ldots, d'_l >$.

The above criteria summarize what the existing research work utilizes to define their negative supporting problems (Lin et al. 2007)(Hsueh et al. 2008)(Ouyang & Huang 2007)(Zheng et al. 2009)(Zheng et al. 2010).

Criteria 2: Negative-First Supporting

Suppose negative elements have higher priority than that of positive ones when verifying whether a data sequence supports a candidate sequence, a negative-first supporting problem needs to be considered as well.

Given a data sequence $s_d = < d_1\ d_2\ \ldots d_m >$ and a candidate sequence $s_c = < c_1\ c_2\ \ldots\ c_n >$, sequence $s'_d = < d'_1\ d'_2\ \ldots\ d'_m > = MES(s_d)$, it is called s_d *supports* s_c if:

- For each negative supper element e_j *(1⩽j⩽l)* in s'_c, there exists integers k_{j_1} and k_{j_2} such that: $< e_j > \sqsubset < d'_{k_{j_1}}, \ldots, d'_{k_{j_2}} >$,

  and (1) *(1⩽k_{i_1}⩽k_{i_2}<k_{j_1}⩽k_{j_2}⩽l)* if e_j has a negative super element e_i right before it.

  and (2) *(1⩽k_{j_1}⩽k_{j_2}<k_{i_1}⩽k_{i_2}⩽l)* if e_j has a negative super element e_i right after it.

- For each positive supper element e_j *(1⩽j⩽l)* in s'_c,

  (1) if *1<j<l*, ∃ integers $k_{(j-1)_2}$, $k_{(j+1)_1}$, $e_j \sqsubseteq < d'_{k_{(j-1)_2+1}}, \ldots, d'_{k_{(j+1)_1-1}} >$.

  (2) if *j=1*, ∃ an integer $k_{(j+1)_1}$, $e_j \sqsubseteq < d'_1, \ldots, d'_{k_{(j+1)_1-1}} >$.

  (3) if *j=l>1*, ∃ an integer $k_{(j-1)_2}$, $e_j \sqsubseteq < d'_{k_{(j-1)_2+1}}, \ldots, d'_l >$.

  (4) if *j=l=1*, $e_i \sqsubseteq < d'_1, \ldots, d'_l >$.

Table 3.4: Examples of the Three Criteria of Supporting

ID	Data Sequence s_d	Support	Candidate s_c	Criteria
E1	$<a\ b\ c\ d>$	$\surd$	$<a\ \neg d\ d>$	Positive-First
E1	$<a\ b\ c\ d>$	$\times$	$<a\ \neg d\ d>$	Negative-First
E1	$<a\ b\ c\ d>$	$\surd$	$<a\ \neg d\ d>$	Order-First
E2	$<a\ b\ c\ d>$	$\times$	$<a\ \neg b\ d>$	Positive-First
E2	$<a\ b\ c\ d>$	$\times$	$<a\ \neg b\ d>$	Negative-First
E2	$<a\ b\ c\ d>$	$\surd$	$<a\ \neg b\ d>$	Order-First
E3	$<a\ b\ c\ d>$	$\times$	$<\neg b\ b\ \neg d>$	Positive-First
E3	$<a\ b\ c\ d>$	$\surd$	$<\neg b\ b\ \neg d>$	Negative-First
E3	$<a\ b\ c\ d>$	$\surd$	$<\neg b\ b\ \neg d>$	Order-First
E4	$<a\ b\ c\ d>$	$\times$	$<\neg a\ c\ \neg c>$	Positive-First
E4	$<a\ b\ c\ d>$	$\surd$	$<\neg a\ c\ \neg c>$	Negative-First
E4	$<a\ b\ c\ d>$	$\surd$	$<\neg a\ c\ \neg c>$	Order-First

Criteria 3: Order-First Supporting

In this criteria, positive and negative elements have the same level of priority. When verifying whether a data sequence supports a candidate sequence, we match all elements by order.

Given a data sequence s_d=$<d_1\ d_2\ \ldots d_m>$ and a candidate sequence s_c=$<c_1\ c_2\ \ldots\ c_n>$, sequence s'_d=$<d'_1\ d'_2\ \ldots\ d'_m>$ is the maximum equivalent sequence of s_d, and sequence s'_c=$<e_1\ e_2\ \ldots\ e_l>$ is a reconstructed candidate sequence. s_d *support* s_c if: for each supper element e_j *(1*$\leqslant j \leqslant$*l)* in s'_c, there exists integers k_{j_1} and k_{j_2} such that: $<e_j> \sqsubset <d'_{k_{j_1}}, \ldots, d'_{k_{j_2}}>$ and *(1*$\leqslant k_{1_1} \leqslant k_{1_2} <\ldots\ < k_{l_1} \leqslant k_{l_2} \leqslant$*m).*

Examples

Table 3.4 shows some examples of the above three criteria of supporting.

In the example E1 in Table 3.4, we find that positive items a and d from s_d contain the corresponding items of s_c. In s_d, item d doesn't occur between

a and d. Therefore, s_d supports s_c in a positive-first problem. For a negative-first problem, we can find a, b, c in s_d to match $\neg d$ of s_c. But we cannot find corresponding items in s_d so as to absolutely contain a and d in s_c, because the reverse partners of a and d can be found in s_d, and this case doesn't belong to absolutely containing.

In the example E2 in Table 3.4, we find positive items a and d from s_d to contain the corresponding items of s_c, but there is an item b occurring between a and d in s_d. Therefore, s_d cannot support s_c in a positive-first problem. For a negative-first problem, we can find a, c, d in s_d to match $\neg b$ of s_c. But we cannot find the corresponding items in s_d to absolutely contain a and d in s_c, because the reverse partners of a and d can be found in s_d.

In the example E3, we find items a and c from s_d to contain the corresponding negative items $\neg b$ and $\neg d$ in s_c. Further, an item b can be found from s_d to absolutely contain b in s_c. Consequently, s_d supports s_c in a negative-first problem. While in a positive-first problem, we do find d after b in s_d, so s_d doesn't support s_c.

In the example E4, s_d doesn't support s_c in a positive-first problem since we can find a before c in s_d, it can't absolutely contain $\neg a$ and c in s_c. While s_d can support s_c for a negative-first problem, since we can find b and d in s_d to contain $\neg a$ and $\neg c$. In addition, c in s_d absolutely contains c in s_c.

In all the examples in Table 3.4, s_d supports s_c for an order-first problem. In the example E1, we find $<a\ b\ c>$ from s_d contains s_c=$<a\ \neg d\ d>$; in E2, $<a\ c\ d>$ from s_d contains s_c=$<a\ \neg b\ d>$; in E3, $<a\ b\ c>$ from s_d contains s_c=$<\neg b\ b\ \neg d>$; in E4, $<b\ c\ d>$ from s_d to contain s_c=$<\neg a\ c\ \neg c>$.

3.4.3 Properties of Negative Supporting

Property: Negative Frequent Property

Based on the above definitions, if we use the order-first criteria to calculate support, the following property follows the Apriori Property (Agrawal et al. 1993):

In an order-first problem, if a negative sequence s *is frequent, all its subse-*

Table 3.5: Apriori-property in a Positive-first Problem

Candidate Sequence	Support	Data Sequence
s_1=<b $\neg c$ a>	×	<b f d c a>
s_2=<b $\neg c$ d a>	√	<b f d c a>

quence must be frequent; or if any subsequence of a sequence s *is not frequent,* s *cannot be frequent.*

Proof: PSP mining follows down-closure property, that is to say, any sub-set of a frequent item-set is also frequent. In the order-first problem of NSP mining, all positive/negative items are treated by the same criteria as they are in PSP mining problem. Therefore, the positive/negative itemset follows down-closure property too. That is to say, if a negative is frequent, all its subsequences are frequent.

If we use the positive-first or negative-first as supporting criteria, the Apriori property doesn't work. Table 3.5 gives out a straight-forward example for the positive-first problem. s_1=<b $\neg c$ a> and s_2=<b $\neg c$ d a> are two candidate sequences and <b f d c a> is a data sequence. The table clearly shows that s does not support s_1, while it supports s_2 even if s_1 is a subsequence of s_2.

3.4.4 Positive/Negative Sequential Pattern

Definition 12: Positive/Negative Sequential Pattern

A candidate sequence s_c is called a *positive/negative sequential pattern* if $sup(s_c) \geqslant min\_sup$, where min_sup is a user-defined support threshold. By contrast, s_c is *infrequent* if $sup(s_c) < min\_sup$.

In order to calculate the support value of a candidate sequence against the data sequences in a sequence database, we need to clarify the criteria of supporting.

Many researchers proposed different kinds of interesting NSP and different kinds of criteria of supporting. To make various criteria to fit into a common

framework, the following two basic operations are proposed. The two basic operations can be used to cover all existing criteria of negative supporting.

3.5 A Framework of NSP Mining

The above sequence containing definition is essential and very general for defining criteria of supporting. In PSP mining, it always considers whether a data sequence supports a positive candidate sequence. But when dealing with NSP mining, we need to consider whether a MES of a data sequence supports a reconstructed negative candidate sequence. We claim that their difference are as follows:

- PSP mining is to mine frequent positive candidate sequences in a sequence database.
- NSP mining is to mine frequent positive/negative candidate sequences in an extend sequence database, which are composed of MES of all data sequences in the original sequence database.

Here we propose a framework of NSP mining as shown in Figure. 3.1. It can be used to describe all the state-of-the-art NSP mining problems.

In this framework, NSP mining is described from three aspects.

(1) Source dataset is extended from an original sequence database by transforming each data sequence to its MES.

(2) Constraints of candidates. As we described in the related work in Section 2, one possible and direct way to use NSP mining is to focus on interesting NSP instead of all of them, since it is impossible to explore all of a huge number of negative candidates. Therefore, many researchers proposed different kinds of interesting NSP algorithms, as described in Section 3.2. Then according to their definitions of interesting NSP, they generate interesting candidates directly or design pruning measures to delete uninteresting candidates.

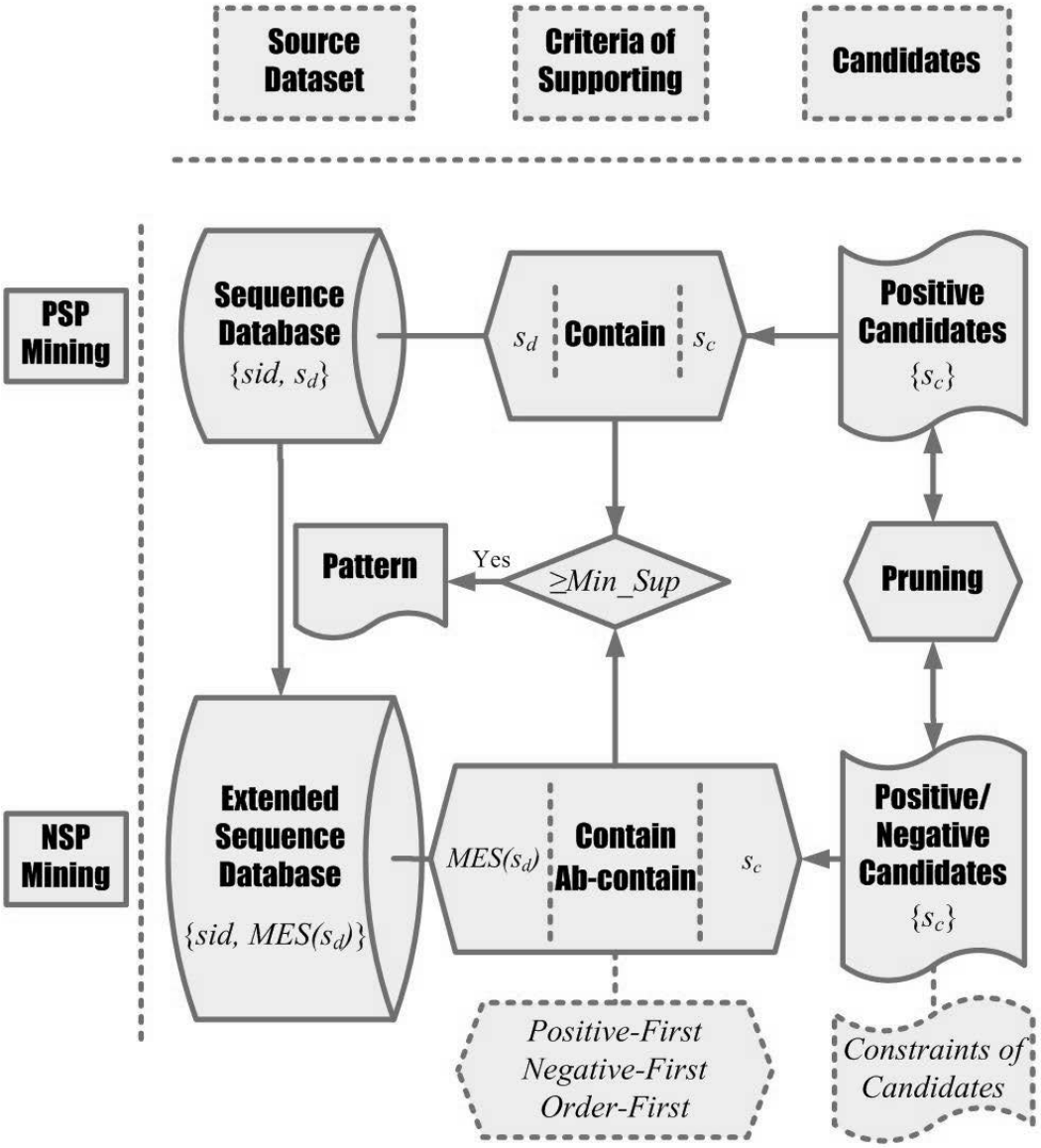


Figure 3.1: A Framework of Negative Sequential Pattern Mining

(3) Criteria of supporting. Current work always gives positive items higher priority than that to negative items. To describe all possible problems of NSP mining, it is necessary to consider negative-first and order-first problems, and we will introduce them in the following section.

Generally speaking, NSP mining has much bigger search space than PSP mining since it needs to search for negative candidate sequences in an extended sequence database, which is much bigger than the original sequence database.

3.5.1 NSP Mining Problem

NSP mining problem is categorized as following three types.

Positive-first Problem

If we use the positive-first criteria to verify whether a data sequence supports a candidate sequence, it is called a *positive-first NSP mining problem*. Usually, it is easy to be understood since positive items indicate those occurred. It is easier for us to understand occurring items than non-occurring ones. Hereby, we illustrate a simple positive-first problem described in Chapter 1 as follows.

Suppose p_1=<a b c d>; p_2=<a $\neg b$ c e>; and each item, a, b, c, d and e, stands for a claim item code in the customer claim database of an health insurance company. The pattern p_1 tells that an insurant usually claims for a, b, c and d in a row. However, the pattern p_2 itself indicates that given an insurant claim for items a and c, if he/she does NOT claim b between a and c, he/she would claim item e instead of d as shown in p_1.

Considering p_2 of the above example, we focus on positive items a, c and e first, and then talk about b which is NOT claimed between a and c, and so that is a positive-first case.

Negative-first Problem

If we use the negative-first criteria to verify whether a data sequence supports a candidate sequence, it is called a *negative-first NSP mining problem*. Different from the positive-first problem, a negative-first problem is hard to imagine since negative items don't occur and it is different from the normal thinking of human being. We thus convert the above example to be a negative-first problem case.

Suppose p_1=<$\neg a$ $\neg b$ $\neg c$ d>; p_2=<$\neg a$ b $\neg c$ e>. By getting the pattern p_1, it tells us that if an insurant does NOT claim for a, b and c in a row, he/she usually claims d after that. However, with the pattern p_2, it indicates that if an insurant does NOT claim for item a or c, but claims b between ¬a and ¬c, then he/she likely claims item e instead of d.

In this example, we focus on negative items $\neg a$ and $\neg c$ in p_2 first, and then talk about b which is claimed between ¬a and ¬c.

Order-first Problem

If we use the order-first criteria to verify whether a data sequence supports a candidate sequence, it is called an *order-first NSP mining problem*.

The above example can also be converted to an order-first problem case.

Suppose p_1=<a b c d>; p_2=<a b $\neg c$ e>. By getting the pattern p_1,it tells that an insurant usually claim for a, b, c and d in a row. However, the pattern p_2 indicates that if an insurant does NOT claim c after claiming items a and b, then next, he/she likely will NOT claim item e but d. Here we see, ordering plays an important role in an order-first problem.

3.5.2 PSP Mining Problem

Positive sequential pattern mining problem is an order-first problem or a positive-first problem.

Proof: In PSP mining, when verifying whether a data sequence contains a candidate, it matches all items of them one by one orderly. Therefore,

PSP mining problem is an order-first problem without considering negative items. And all items in PSP mining are positive items, that is to say, it is a positive-first problem as well.

3.6 NSP Mining Problem in the Thesis

3.6.1 Constraints on Interesting NSP

As we described in the literature review of Chapter 2, one possible and direct way to solve NSP mining is to focus on interesting NSP instead of all NSP, since it is impossible to explore a huge number of negative candidates. Therefore, many researchers proposed different kinds of interesting NSP.

(Ouyang & Huang 2007) proposed NSP in the formats of $<A\ \neg B>$, $<\neg A\ B>$ and $<\neg A\ \neg B>$, where A and B are frequent subsequences and $A \cap B = \varnothing$. It required $<A\ B>$ to be frequent if $<A\ \neg B>$ is NSP. And also, it set a threshold *min_interest* for interesting NSP. The final definition follows three constraints. $<A\ \neg B>$ is an interesting NSP, if (1). $A \cap B = \varnothing$; (2). $\text{sup}(A) \geq min\_sup$, $sup(B) \geq min\_sup$, $sup(A \cup \neg B) \geq min\_sup$; (3). $sup(A \cup B) - sup(A) \times sup(\neg B) \geq min\_interest$.

(Lin et al. 2007) only focused on NSP which has a negative item at the end of a pattern, like $<c_1,\ c_2,\ \ldots,\ \neg c_n>$, where every c_i, c_i $(1 \leqslant i \leqslant n)$ is positive item. Such constraint is very strong for shrinking the search space.

In the literature, (Hsueh et al. 2008) proposed the most similar definitions of NSP as ours, it stated that meaningful NSP should follow some constraints. The first one is the *size constraint*, that is to say, a NSP can't be supported by a data sequence, when the length of the data sequence is shorter than that of the NSP, or smaller in size. For example, given $s_d = <a\ b>$, $s_{c_1} = <a\ \neg c>$ and $s_{c_2} = <a\ \neg c\ d>$, s_d can support s_{c_1}, but can not support s_{c_2}, since the size of s_{c_2} is 3 and s_d is only 2. The second constraint is the *frequency constraint*. For each negative item in a negative pattern, it required its corresponding positive item to be frequent too. For example, $\neg a$ is a valid negative item if a is frequent. The third constraint is the *formation constraint*, which doesn't

allow two or more negative itemsets occur adjacently.

Our works target at mining general NSP without too strong constraints. Therefore, we only define following three constraints for the proposed methods.

- For each negative item in a negative pattern, its positive item is required to be frequent. For example, if *(¬c)* is a negative item, its positive item *(c)* is required to be frequent. It is helpful for us to focus on the frequent items.

- Two or more than two adjacent negative elements are not accepted in the negative sequence, since it is hard to be explained in real-world applications. This constraint is also used by other researchers (Hsueh et al. 2008).

- Items in a single element should be either all positive or all negative. For example, $<a\ (a\neg b)\ c>$ is not allowed since items a and $\neg b$ occur in the same element.

3.6.2 Criteria of Negative Supporting

Different Definitions in Related Work

All existing work focuses on the positive-first NSP mining problem, and even with that, different researchers still present inconsistent definitions and explanations since they represent different interesting negative patterns. (Hsueh et al. 2008) considers that data sequence s_d=$<d\ c>$ cannot contain negative candidate s_c=$<(\neg a\neg b)\ c\ \neg d>$ since *size(s_c)>size(s_d)*. They also add more constraints in the support calculation, where they defined *n-cover* and *n-contain* to describe how a candidate matches a data sequence. Another issue is how to deal with a non-occurring element. (Hsueh et al. 2008) argues that s_d=$<d\ c>$ cannot contain $<\neg c\ d>$ because $<d>$ in s_d has no antecedent itemset; and s_d cannot contain $<c\ \neg d>$ because $<c>$ in s_d has no successor. Furthermore, the containment position of each element is very tricky. (Hsueh

Table 3.6: Neg-GSP: Example of Negative Supporting

Candidate Sequence	Supporting	Data Sequence
$<b\ \neg c\ a>$	$\surd$	$<b\ d\ a>$
$<b\ \neg c\ a>$	$\surd$	$<b\ d\ a\ c>$
$<b\ \neg c\ a>$	$\times$	$<b\ d\ c\ a>$

et al. 2008) proposes that a data sequence s_d=$<a\ a\ c\ b\ c>$ cannot contain a negative candidate s_c=$<a\ \neg b\ c>$, since s_d also has a subsequence $<a\ b\ c>$, which can be an opposite evidence of s_c.

Negative Supporting of this Thesis

The above discussions show that there is not a consolidated concept of negative supporting in the literature. We defined a more consistent definition as follows.

A data sequence s_d=$<d_1\ d_2\ ...\ d_m>$ supports a negative candidate s_c=$<e_1\ e_2\ ...\ e_k>$, if:

1) s_d contains the max positive subsequence of s_c

2) for each negative element e_i $(1 \leqslant i \leqslant k)$, there exist integers p_i, q_i, r_i $(1 \leqslant p_i \leqslant q_i \leqslant r_i \leqslant m)$ such that: $\exists e_{i-1} \subseteq d_{p_i} \wedge e_{i+1} \subseteq d_{r_i}$, and for $\forall d_{q_i}$, $e_i \not\subset d_{q_i}$; and $(1 \leqslant p_1 \leqslant q_1 \leqslant r_1 \leqslant \ldots p_k \leqslant q_k \leqslant r_k \leqslant m)$

Table 3.6 gives examples of negative supporting, s_c=$<b\ \neg c\ a>$ is supported by $<b\ d\ a\ c>$, but is not supported by $<b\ d\ c\ a>$, since the negative element c appears between the element b and a.

The above supporting definition also belongs to the positive-first problem.

3.7 Conclusions

- The search space of NSP mining problem is quite enormous. The number of candidates could be much larger than that of PSP mining. Therefore, it is important to target on the most interesting candidates, which follows some predefined constraints.

- The NSP mining problem can be categorized as three types, positive-first, negative-first and order-first problems.
- Positive sequential pattern mining problem is an order-first and positive-first problem.

Chapter 4

Neg-GSP Algorithm

4.1 GSP Algorithm

4.1.1 General Description of GSP

GSP algorithm (Srikant & Agrawal 1996) is a classical and widely accepted algorithm for sequential pattern mining. It makes multiple passes over a sequence database to find sequential patterns. The first pass starts from calculating the support of every 1-item candidate. At the end of the first pass, all of the 1-item patterns, which are then used as seeds to generate new candidates for the next pass, are obtained. Each new candidate of the next pass has one more item than its seeds. Then, the candidates are filtered by pruning to remove infrequent ones. After pruning, the supports of the new candidates are counted by another passing over the sequence database, and frequent patterns become the seeds for the next pass. The algorithm terminates when there is no more frequent pattern at the end of a pass, or when no candidate is generated.

Three essential steps of GSP are as follows:

Step 1. Generating candidates. Candidate sequences are generated by joining, and the algorithm tries to generate candidates as few as possible.

Step 2. Pruning candidate. To prune potential infrequent candidates as

many as possible.

Step 3. Counting candidates. This step is to calculate support value of each candidate sequence. If the value of a candidate is greater than the predefined threshold *min_sup*, the candidate is put to the patterns set.

4.1.2 Generating Candidates

Each candidate is generated by the joining step described below. Let L_k denote the set of all frequent k-item sequences, and C_k is the set of k-item candidate sequences.

We generate candidates by joining all frequent (k-1)-item patterns in L_{k-1}. A sequence s_1 joins with s_2 if the subsequence obtained by dropping the first item of s_1 is the same as that obtained by dropping the last item of s_2. By joining s_1 with s_2, a new candidate sequence is generated. It is the sequence s_1 extended with the last item in s_2. The added item becomes a separate element on the new candidate if it is a separate element in s_2, or becomes part of the last element if it is part of the last element of s_2. When joining L_1 with L_1, we need to add the item of s_2 both as a part of an element and as a separate element of the new candidate.

For example, joining two 1-item sequences <*a*> and <*b*> produces three 2-item candidate sequences: <*a b*>, <*b a*> and <*(ab)*>. Joining two 4-item sequences <*a (bc) d*> and <*(bc) d e*> produces <*a (bc) d e*>. Joining two 4-item sequences <*a (bc) d*> and <*(bc) (de)*> produces <*a (bc) (de)*>.

4.1.3 Pruning Candidates

We delete a k-item candidate sequence if it has at least one (k-1)-item subsequence whose support count is less than the threshold *min_sup*.

For example, if <*(ab) d*>, <*b (ad)*> and <*b (de)*> are in a 3-item patterns set, joining <*(ab) d*> and <*b (de)*> will generate a 4-item candidate <*(ab) (de)*>, which has 3-item subsequences including <*b (de)*>, <*a (de)*>, <*(ab) e*> and <*(ab) d*>. Since two of them, <*a (de)*> and <*(ab) e*>, are not in

the 3-item patterns set, <*(ab) (de)*> can be pruned.

4.1.4 Counting Candidates

While passing over a sequence database to get the support values of all candidates, we read one data sequence at a time and increase the support counts of the candidates contained in the data sequence. Thus, given a set of candidate sequences C and a data sequence s_d, we need to find all sequences in C that are contained in s_d. Two techniques are utilized to solve this problem:

1. It adds candidate sequences into a hash-tree to reduce the number of candidates in C that need to be checked for a data sequence.

2. It transforms the format of a data sequence so that it can efficiently find whether a data sequence contains a specific candidate sequence.

4.1.5 Procedure of GSP

The pseudo-code of GSP are given as follows.

GSP-Algorithm

Function: To find frequent sequential patterns in a sequence database.
Input: Sequence Database D_s, frequency threshold *min_sup*.
Output: Frequent Sequential Patterns L=L_1, L_2, ..., L_m.

RUN_GSP_Algorithm(D_s, *min_sup*){
01. Take sequences in form of <x> as 1-item candidates
02. Passing over sequence database to find 1-item patterns set L_1.
03. k=1;
04. **while** (L_k is not empty) {

```
05.        C_{k+1} = Join(L_k, L_k);
           ▷ Generate (k+1)-item candidates set C_{k+1} from L_k;
06.        C_{k+1} = Prune(C_{k+1});
07.        if C_{k+1} is not empty {
08.           for (each s_c in C_{k+1}) {
              ▷scan database once, to find (k+1)-item patterns set;
09.              if (support(s_c) >= min_sup)
10.                 L_{k+1}.add(s_c);
11.           }
12.        }
13.        k=k+1;
14.        L.add(L_{k+1});
15.     }
16.     return L;
}
```

4.1.6 Improving GSP to Find NSP

The NSP mining problems of the thesis, including the constraints of candidates and supporting criteria, have been described in Section 3.6.

Based on GSP algorithm, we need to adapt and modify the joining and pruning steps for negative sequences to ensure searching the space integrality, and reduce the number of negative candidates as well.

If the joining step only happens in the *(k-1)-item* patterns set, some *k-item* candidates could be missed. Therefore, we generate a (k-1)-item seed set $S_{(k-1)}$, which is a super set of (k-1)-item patterns set L_{k-1}, to join and generate new candidates, and so we can ensure to cover all potential negative candidates.

We use the following measure, base-support, to verify whether a candidate is a potential seed.

Table 4.1: Examples of base-support and support

Data Sequence S_d	Base-Support	Support	Candidate
$< b\ d\ c\ a >$	$\surd$	$\times$	$< b\ \neg c\ a >$
$< b\ d\ a >$	$\surd$	$\surd$	$< b\ \neg c\ a >$

Definition: Base-support

A data sequence s_d *base-supports* a candidate s_c if s_d **contains** s_c, see Section 3.4.1. That is to say, a data sequence s_d *base-supports* a negative candidate s_c if s_c is a subsequence of $MES(s_d)$.

For example, a data sequence s_d=$<b\ d\ a>$ base-supports a candidate sequence s_c=$<b\ \neg c\ a>$. A data sequence s'_d=$<b\ d\ c\ a>$ base-supports s_c as well, since the element d in s'_d, which contains $\neg c$, can be found between the elements b and a. See table 4.1.

If a data sequence base-supports a candidate sequence, then its *base_support* value is to be increased. The value is used to verify whether a candidate is a potential seed. If the base_support of a candidate is greater than *min_sup*, it is a potential seed, otherwise, it cannot be a seed. The set of k-item seeds with *base_support*$\geqslant$*min_sup* can ensure that it won't miss any k-item candidates by joining in them.

This measure is efficient and easily to be proved. Base-support follows the order-first matching, and the Apriori property works in an order-first problem as we described in the negative frequent property, see Section 3.4.3, therefore, it won't miss any potential candidates.

4.2 Process of Neg-GSP

Suppose D is a sequence database, D=$\{s_{d_1},\ s_{d_2},\ s_{d_3},\ ...,\ s_{d_n}\}$, where s_{d_i} *(1*$\leqslant i \leqslant$*n)* is a data sequence. Then the objective of NSP mining is to find frequent NSP in D, with a minimum support threshold *min_sup*. We describe the process of Neg-GSP as follows.

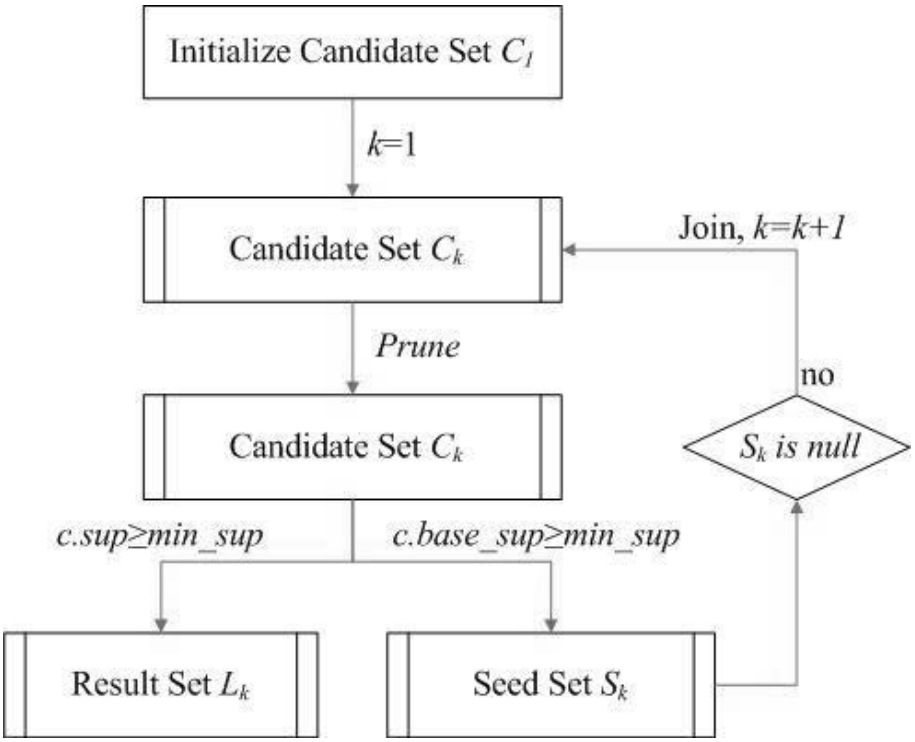


Figure 4.1: The Process Flow of Neg-GSP

Firstly, we utilize GSP algorithm to generate all PSP. Assume L_{pos}=$\{L_{pos,1}$, $L_{pos,2}$, ..., $L_{pos,l}\}$, where $L_{pos,j}$ $(1 \leqslant j \leqslant l)$ represents a j-item frequent PSP set.

Next, we begin to generate NSC from L_{pos}. We transform all 1-item PSP in $L_{pos,1}$ to their corresponding 1-item negative sequences, which are taken as the initial 1-item negative candidates set $C_{neg,1}$.

After those initial processes, we can start to mine NSP. A process flow of Neg-GSP is also demonstrated in Figure 4.1.

For each 1-item candidate in $C_{neg,1}$, if its $base\_support \geqslant min\_sup$, it is added into 1-item seed set $S_{neg,1}$. After that, we get a processed 1-item seed set $S_{neg,1}$, which is then used to generate a 2-item negative candidate set $C_{neg,2}$. For each candidate in $C_{neg,1}$, if its support is higher than min_sup, it is added into 1-item frequent patterns set $L_{neg,1}$.

By joining k-item seed set $S_{neg,k}$, it produces a (k+1)-item candidates set $C_{neg,k+1}$. The new candidates set may include many invalid candidates, there-

Table 4.2: An Example of Joining

c_i	e_{i_1}	e_{i_2}	e_{i_3}	...	$e_{i_{k-1}}$	e_{i_k}	
		=	=	=	=	=	
c_j		e_{j_1}	e_{j_2}	...	$e_{j_{k-2}}$	$e_{j_{k-1}}$	e_{j_k}
$\Rightarrow c_{k+1}$	e_{i_1}	e_{i_2}	e_{i_3}	...	$e_{i_{k-1}}$	e_{i_k}	e_{j_k}

fore, pruning invalid candidates is necessary and helpful for further search. And an idea of pruning is to verify whether the maximum positive subsequence of a candidate is frequent. After pruning, invalid candidates are pruned, and valid ones are kept in the candidate set. Then, by passing over the sequence database D, we get the support values of all (k+1)-item candidates. Again, the (k+1)-item candidates with support higher than min_sup are output to the patterns set $L_{neg,k+1}$.

After the above procedures are performed, we get a (k+1)-item result set $L_{neg,k+1}$ and a (k+1)-item seed set $S_{neg,k+1}$ for next iteration. Longer patterns are generated by repeating the above process until the candidate set is empty. Each iteration will generate and output frequent NSP into L_{neg}=$\{L_{neg,1}, L_{neg,2}, ..., L_{neg,l}\}$, which is the final results set.

4.3 Neg-GSP Algorithm

4.3.1 Joining to Generate Candidates

The (k+1)-item candidates are generated by joining k-item seeds. Given two k-item seed sequences c_i=<e_{i_1} e_{i_2} e_{i_3} ... $e_{i_{k-1}}$ e_{i_k}> and c_j=<e_{j_1} e_{j_2} e_{j_3} ... $e_{j_{k-1}}$ e_{j_k}>. c_i' is a sequence by deleting the first item of c_i and c_j' is a sequence by deleting the last item of c_j. If c_i'=c_j', c_i and c_j can join to generate a new (k+1)-item candidate c_{k+1}, which is composed of c_i' and the last item of c_j. If the item is part of the last element of c_j, it will still be part of last element of c_{k+1}; if the item is a separate element of c_j, then it is still to be a separate element of c_{k+1}, see Table 4.2.

The above joining method is similar to that of GSP. But there is a difference while generating the seed set. GSP algorithm uses the patterns set as the seed set, and generates all (k+1)-item candidates by joining the k-item patterns set. However, Apriori property doesn't work in the NSP mining problem, and we need to join not only k-item patterns but also some infrequent k-item sequences, as we described in the above.

While performing the joining operation, we don't join positive patterns with themselves, since that has been done in the PSP mining at the first step of Neg-GSP algorithm.

4.3.2 Pruning Invalid Candidates

In NSP mining, the number of candidates are much more than those in PSP mining. It is important to design effective pruning methods to prune invalid candidates, which were generated by joining step but were not possible to be frequent patterns, from the candidates set.

While pruning happens in k-item candidates with GSP algorithm, it prunes the candidates with at least one (k-1)-item infrequent subsequence. However, the above pruning method won't work for NSP mining. For example, given a candidate s_c=<a b $\neg c$ d $\neg e$>, s'_c=<a b $\neg c$ $\neg e$> is a subsequence of s_c and is infrequent. As a result, s_c should be pruned since its subsequence s'_c is not frequent. But in fact, s_c still can be frequent. Another problem is that we don't allow two adjacent negative elements in a negative sequence, s'_c is an invalid negative sequence. According to the above two points, the pruning method used by GSP algorithm can not be used in the NSP mining problem directly.

We propose a pruning method below. Suppose s'_c is the maximum positive subsequence of a candidate s_c. If s'_c is not frequent, s_c must be infrequent and should be pruned. This method is simple but effective to prune invalid candidates without missing potential valid candidates.

4.3.3 Generating Seed Set for Next Pass

Given an infrequent 3-item sequence $<b\ \neg ca>$, its 4-item candidate $<b\ \neg c\ d\ a>$ may still be frequent. For detailed explanation, please refer to Section 3.4.3 and the example in Table 3.5. Therefore we need to count $<b\ \neg c>$ as a seed for joining and generating 3-item candidate $<b\ \neg c\ d>$.

A k-item sequence is regarded as a k-item seed sequence if its *base_support* is greater than *min_sup*. Therefore, the candidates with *base_support* greater than *min_sup* are added into the seed set. Otherwise, it can not be used to generate any (k+1)-item frequent pattern.

4.3.4 Algorithm Description

The proposed algorithm Neg-GSP is described as follows.

Step 1: Find all PSP by the traditional GSP algorithm (Srikant & Agrawal 1996).

Step 2: Transform 1-item positive patterns to 1-item negative candidates, and then get 1-item seed set and 1-item patterns.

Step 3: For all (k-1)-item seeds, perform the joining operation with each other and generates k-item candidates. (k-1)-item positive patterns and (k-1)-item seed sequences are joined since it can generate valid k-item negative candidates as well.

Step 4: Prune unnecessary candidates to get a smaller candidate set.

Step 5: Count *support* and *base_support* of all candidates.

Step 6: For each candidate, if its *base_support* is greater than *min_sup*, it is added to k-item seed set. If its *support* is greater than *min_sup*, then it is frequent and outputted as a k-item pattern.

Step 7: If k-item seed set is not empty, increase k by one and loop back to Step 3 until the next candidate set is empty.

The procedure is illustrated with an example as Figure 4.2.

Neg-GSP Algorithm

Function: Find negative sequential patterns from a sequence database.
Input: Sequence database D, *min_sup*
Output: NSP set L

RUN(D, *min_sup*) {
▷ /∗ S: Seeds set; C: Candidates set; L: Results set; ∗/
01. k = 1;
02. C_k = initialize();
03. $S_k = C_k$;
04. **while** (S_k.size() > 0){
05. C_{k+1} = Join(S_k);
06. C_{k+1} = Prune(C_{k+1});
07. **for** (each c in C_{k+1}){
08. **if** (c.base_support > *min_sup*) S_{k+1}.add(c);
09. **if** (c.support > *min_sup*) L_{k+1}.add(c);
10. }
11. L.add(L_{k+1});
12. k = k + 1;
13. }
14. **return** L;
}

4.3.5 Computational Complexity Analysis

Dong etc,. (Dong 2009) stated that the problem of mining the complete set of sequential patterns is #P-complete theoretically. Therefore, it is impossible to have a polynomial time algorithm unless P=NP. Even if P=NP, it is still unclear whether a polynomial time algorithm exists.

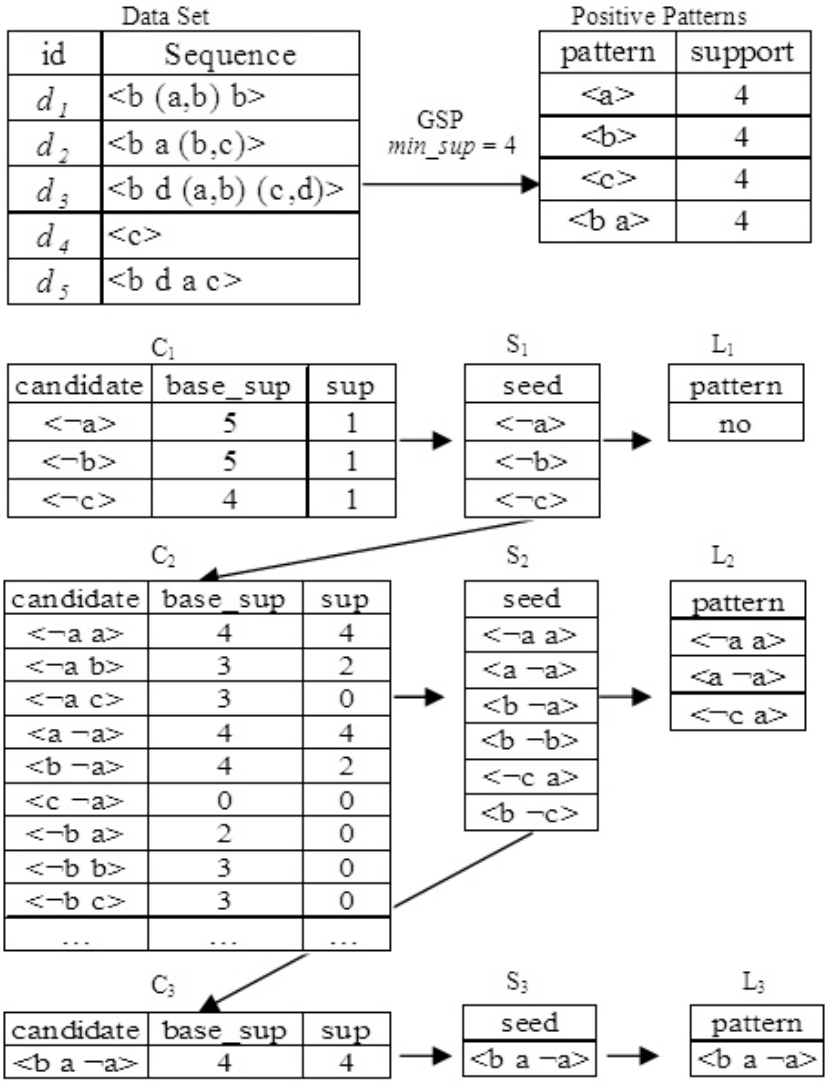


Figure 4.2: Neg-GSP: An Example

Neg-GSP and PNSP will scan database once for every canidate to get its support value. Therefore, the count of candidates will affect the computational complexity. PNSP and Neg-GSP are both Apriori-based algorithms. We will analyze their computational complexity in the following part.

Let's assume that the number of negative candidates N is:

$$N = \sum_{k=1}^{m} n_k$$

where m is the length of the longest pattern, and n_k is the number of k-item candidates.

Neg-GSP: Neg-GSP generates 2-item candidates by joining 1-item patterns. In the worst case, the number of 1-item patterns equals to the number of 1-item candidates, that means all 1-item candidates are frequent. Let's assume that all 1-item patterns are able to join with each other to generate 2-item candidates, and then the algorithm will generate n_2 2-item candidates, $n_2 = n_1^2$. Therefore, we can get that $n_{k+1} = n_k^2$.

$$\begin{aligned} N &= n_1 + n_2 + n_3 + \ldots + n_m \\ &= n_1 + n_1^2 + n_2^2 + \ldots + n_{m-1}^2 \\ &= n_1 + n_1^2 + n_1^4 + \ldots + n_1^{(m-1)*2} \end{aligned}$$

Based on the above, the computational complexity of Neg-GSP is $O(n_1^m)$, where n_1 is the number of 1-item patterns, and m is the length of the longest pattern.

PNSP: PNSP generates (k+1)-item candidates by appending 1-item patterns to the end of k-item patterns. In the worst case,

$$\begin{aligned} n_2 &= n_1 * n_1 = n_1^2 \\ n_3 &= n_2 * n_1 = n_1^3 \\ n_4 &= n_3 * n_1 = n_1^4 \\ &\ldots \\ n_m &= n_{m-1} * n_1 = n_1^m \end{aligned}$$

Since $N = \sum_{k=1}^{m} n_k$, the computational complexity of PNSP is $O(n_1^m)$ too,

where n_1 is the number of 1-item patterns, and m is the length of the longest pattern.

Therefore, Neg-GSP and PNSP have a same level computational complexity. But in practise, joining operation of Neg-GSP is much more effective than that of PNSP, because PNSP appends 1-item patterns into the end of k-item patterns to generate (k+1)-item candidates in a very simple way.

4.4 Experiments

4.4.1 Datasets

Two synthetic datasets are generated by IBM data generator (Agrawal & Srikant 1995), and are used to test the proposed algorithm in the experiments.

$Dataset1(DS1)$ is C8.T8.S4.I8.DB10k.N1k. It contains 10k sequences, the number of items is 1000, the average number of elements in a sequence is 8, the average number of items in an element is 8, average length of maximal pattern consists of 4 elements and each element is composed of 8 items averagely. The minimum number of elements in a sequence is 1, and the maximum number is 237.

$Dataset2(DS2)$ is C10.T2.5.S4.I2.5.DB100k.N10k. It contains 100k sequences, the number of items is 10k, the average number of elements in a sequence is 10, the average number of items in an element is 2.5, average length of maximal pattern consists of 4 elements and each element is composed of 2.5 items averagely. The minimum number of elements in a sequence is 1, and the maximum number is 138.

Features of the two synthetic datasets are listed in Table 4.4.1.

Data set 3 (DS3) is real application data for insurance claims. The data set contains 479 sequences. The average number of elements in a sequence is 30. The minimum number of elements in a sequence is 1, and the maximum number is 171.

Table 4.3: Neg-GSP: Features of Synthetic Datasets

Parameters	*DS1*	*DS2*
Number of sequences (DB)	10k	100k
Number of items (N)	1k	10k
Average number of elements per sequence (C)	8	10
Average number of items per element (T)	8	2.5
Average length of maximal potentially large sequences (S)	4	4
Average size of itemsets in maximal potentially large sequences (I)	8	2.5

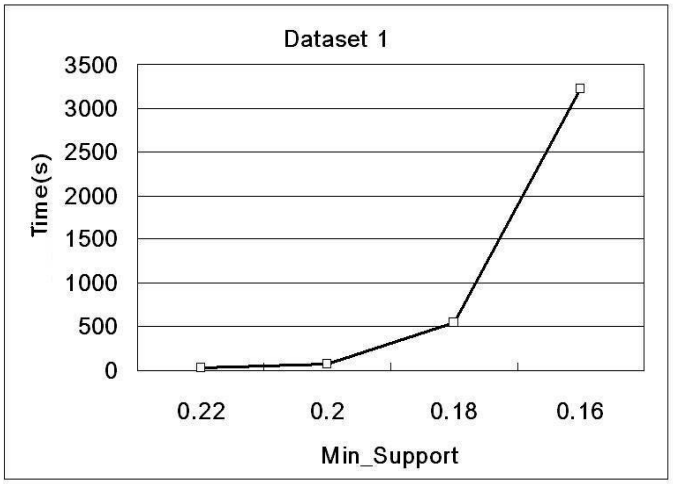
4.4.2 Performance Evaluation

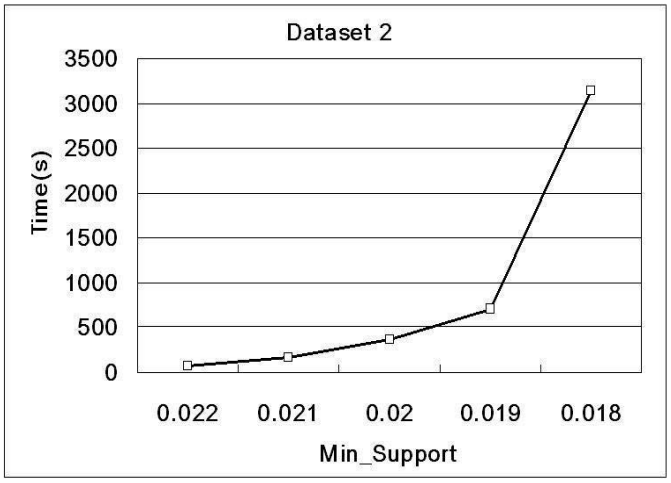
The proposed algorithm Neg-GSP was implemented with Java and tested on a PC with Intel Core 2 CPU of 2.9GHz, 2GB memory and Windows XP Professional SP2.

We compared the execution time of Neg-GSP on different support thresholds, see Figure 4.3, and compared counts of patterns on different support thresholds as well, see Figure 4.4. Negative pattern mining costs much more execution time than positive pattern mining because the candidates counts are not of the same magnitude, especially when the support threshold is set very low.

4.4.3 Comparison with PNSP Algorithm

Comparing Neg-GSP algorithm with PNSP algorithm (Hsueh et al. 2008), the results of execution time (see Figure 4.5) show that Neg-GSP outperforms PNSP in terms of execution time. The reason is that PNSP generates more invalid candidates. Its negative candidates may increase sharply when the number of negative frequent 1-item patterns increases. When there are a huge number of negative candidates, Neg-GSP will cost lots of execution time in the joining process. That would degrade its performance. Therefore,





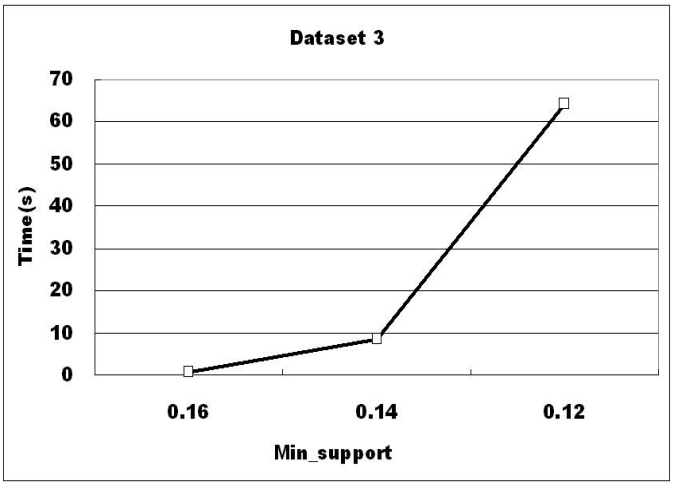


Figure 4.3: Neg-GSP: Execution Time

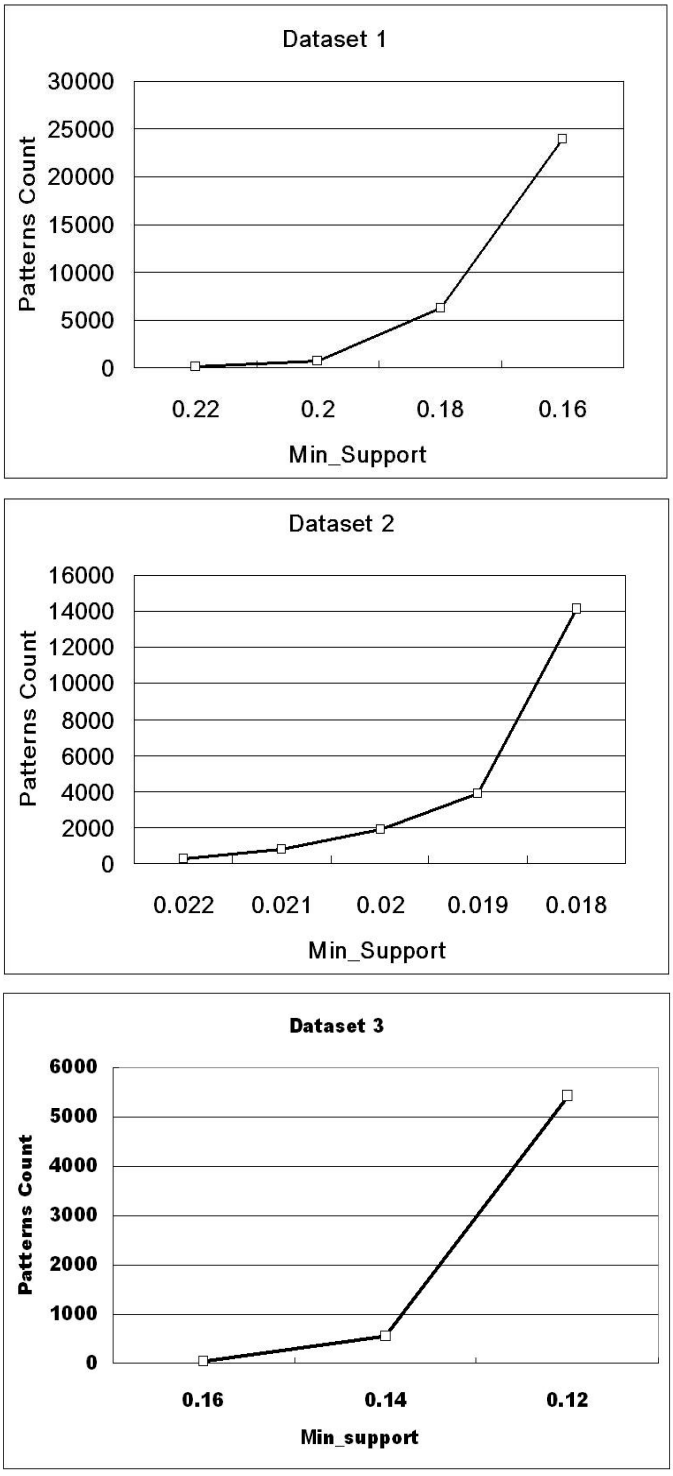


Figure 4.4: Neg-GSP: Patterns Counts

when *min_sup* is set very low, Neg-GSP can not have performance as good as the case when *min_sup* is high.

4.5 Conclusions

In this chapter, we proposed a NSP mining method, Neg-GSP, based on the classical GSP algorithm.

- Joining and pruning are two essential steps of GSP, they are adapted to negative mining in Neg-GSP.
- An effective pruning method is proposed to reduce the number of candidates by pruning invalid candidates.
- The efficiency and effectiveness of the proposed algorithm shown in the experiments, which are tested on two synthetic datasets and a real-application dataset, outperforms another existing NSP mining algorithm PNSP.
- Neg-GSP has a same computational complexity as PNSP, but in practise, Neg-GSP can generate candidates more efficiently than PNSP.

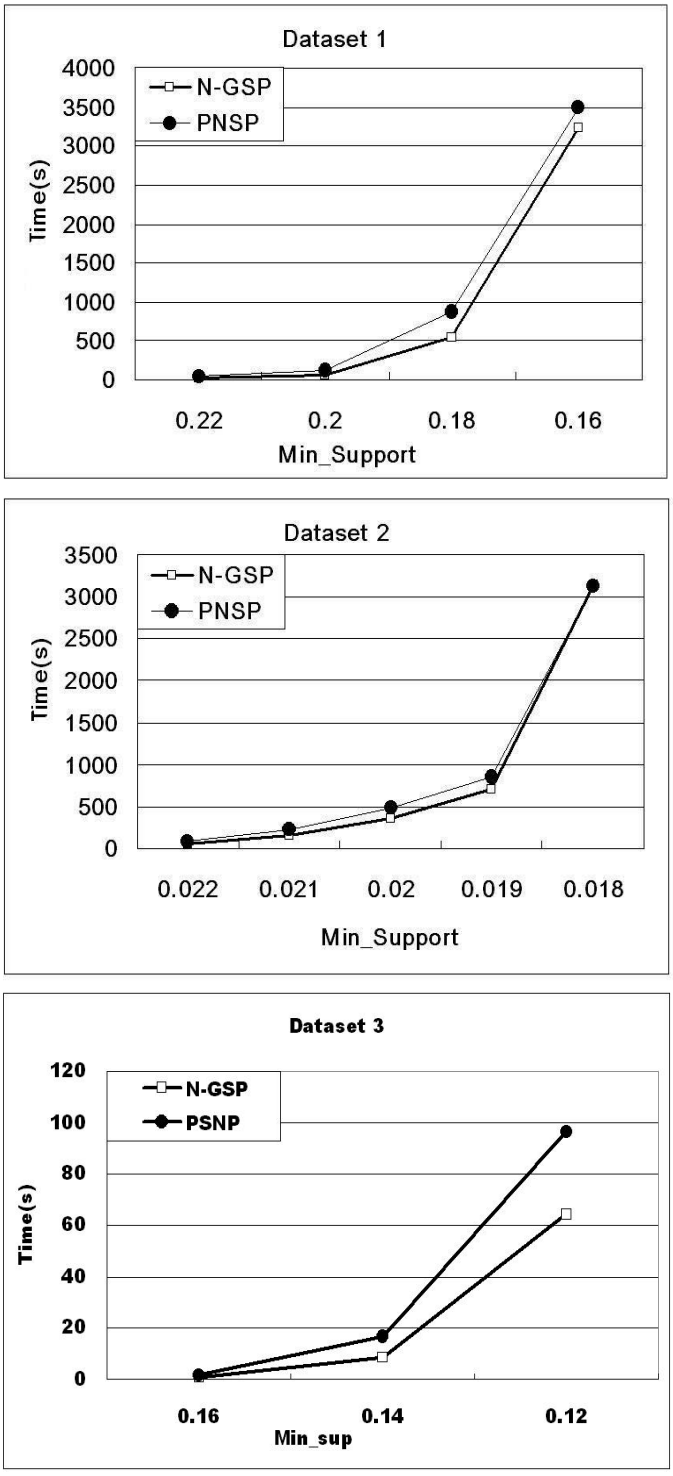


Figure 4.5: Neg-GSP: Comparison with PNSP Algorithm

Chapter 5

Genetic Algorithm Based Algorithm: GA-NSP

Since Neg-GSP generates huge amount of candidates, it will cost much execution time on calculating support values of all candidates by scanning sequence database again and again. It is still quite hard to improve the performance. Since evolution algorithms, such as Genetic Algorithm, are able to find the best solutions/patterns by evolution. Better or more frequent sequences can be kept for generating new offsprings with higher priority. That would make it be more effective to mine PSP or NSP. Based on this idea, a Genetic Algorithm based method is proposed, as GA-NSP.

First of all, a brief introduction of Genetic Algorithm will be given as following.

5.1 Genetic Algorithm

Genetic Algorithm simulates biological evolution and genetic mechanism of natural selection process. It can be used to search the best solution by simulating natural evolutionary process. Genetic Algorithm was firstly proposed by J. Holland in 1960. In the book (Holland 1975), it presented Genetic Algorithm in a general theoretical framework. After that, research on Genetic

Algorithm has spread quickly in computer science.

Genetic Algorithm is a population-based model, it uses three basic operators, including selection, crossover and mutation, to generate new solution in a search space. Many Genetic Algorithm models have been introduced by researchers largely working from an experimental perspective. Many of these researchers are application oriented and are typically interested in Genetic Algorithm as optimization tools. (Whitley 1994)

Genetic Algorithm allows a population composed of many individuals to evolve under specified rules. It starts from a randomly selected individuals set named population. This population is used to produce a next generation of individuals by reproduction, which includes crossover and mutate operation. When getting a new generation of individuals, that is a new population, it evaluates the new individuals' fitness. Then it selects new individuals with high fitness from the population to reproduce for next generation. Individuals with high fitness have more chance to be selected.

5.1.1 Procedure of Genetic Algorithm

First of all, to utilize Genetic Algorithm, encoding a problem to potential solution is essential. For example, the solution could be as a string of numbers, a binary bit string, or any other kinds of chromosome code. A typical potential solution (chromosome) may look like following:

01111011001011001011110111

After that, an initial population, which is composed of many chromosomes, is created, and it is usually created randomly. Each chromosome in the population always represents a solution to the problem to be solved. Then, following steps are repeated to find the best solution from the population. The population evolutes all the time and keeps good genes in it.

(1) Test each chromosome to see how good it is at solving the problem and assign a fitness score accordingly. The fitness score is a measure of how

good the chromosome is at solving the target problem.

(2) Select two chromosomes from the current population. The chance of being selected is proportional to the chromosomes fitness. Roulette wheel selection is a commonly adopted method.

(3) Based on a predefined crossover rate, crossover the selected chromosomes and generate next generation of chromosomes.

(4) Based on a predefined mutation rate, change the chosen chromosomes at a randomly chosen point to generate a new chromosome.

(5) Repeat step 2, 3, 4 until the best solution has been found, or maximum number of generations are reached.

The basic algorithm is as below:

Basic Genetic Algorithm

Function: Basic process of Genetic Algorithm.

Input: Maximum Generation Count $max\_generation$, Crossover rate threshold $Crossover\_Rate$, mutation rate threshold $Mutation\_Rate$.

Output: Best individual(s)

GeneticAlgorithm($max\_generation$){

01. Initialize population;
02. Evaluate fitness of individuals in initial population;
03. int $genCount$=0;
04. **while** ($genCount < max\_generation$){
05. Selection; ▷select individuals for reproduction;
06. **if** (RandomRate < $Crossover\_Rate$) Crossover();
07. **if** (RandomRate < $Mutation\_Rate$) Mutation();
08. Evaluate fitnesses of individuals in new population;
09. $genCount$++;

Table 5.1: Genetic Algorithm: Examples of Encoding

Encoding Methods	**Examples**
Binary Encoding	011110110010
Permutation Encoding	4 5 8 1 43 13 4 12
Value Encoding	1.32 4.2 99.2 123.2 3.123 5.2
Tree Encoding	$(- \ A \ (\ \div \ 15 \ B \ ) \ )$

10. }
11. **return** best_individual(s).

}

5.1.2 Encoding

Encoding of chromosomes is to define the problem to be solved. It enables the problem's solutions to be represented by chromosome. Encoding depends heavily on the needs of the problem to be solved.

There are many different methods of encoding chromosome, such as binary encoding, permutation encoding, value encoding and tree encoding. Binary encoding is used the most widely. Permutation encoding method is suitable for ordering problem. Value encoding can be used for complicated value, such as real numbers. Some examples are given by Table.5.1 for a straightforward demonstration.

5.1.3 Fitness Function

In order to evaluate the chromosomes/individuals and decide which are the best ones for next generation, fitness function is implemented in Genetic Algorithm. The function is to find good solution for the problem to be solved.

The selection function is to use the fitness values to select the parents of the next generation. Individuals with higher fitness value will have more

chance to be selected as parents.

Fitness function affects the performance of the Genetic Algorithm. If fitness values of population are too widely, the individuals with the highest values reproduce very rapidly, and preventing the algorithm from searching other areas of the solution space. On the other hand, if the values vary only a little, all individuals have approximately the same chance of reproduction and the search will progress very slowly. (MathWorks 2011)

5.1.4 Selection

The selection function chooses parents from population for the next generation according to their fitness.

Roulette wheel selection (Bäck 1996), also called fitness proportionate selection, is one of the most popular selection methods. In roulette wheel selection, individuals are given probabilities of being selected that are directly proportionate to their fitness. An individual could be selected more than once as a parent, in which case it contributes its genes to more than one child. Two parents are then chosen and produce offspring.

Some other selection methods include stochastic universal sampling (Baker 1987), tournament selection (Miller & Goldberg n.d.), truncation selection and so on.

5.1.5 Crossover

Crossover is an operator that is used to make chosen chromosomes to generate new generations. There are many different kinds of crossover methods. Single point crossover, two-point crossover, cut and spice and so on.

Single Point Crossover

The most common method is single point crossover. Single point crossover is to swap data between the two parent chromosomes at a predefined single point. Table.5.2 shows an example of single point crossover.

Table 5.2: Genetic Algorithm: Single Point Crossover

*parent*1	0111 ↕ 10110010	⇒	*child*1	0111 **00111001**
*parent*2	**1100** ↕ **00111001**	⇒	*child*2	**1100** 10110010

Table 5.3: Genetic Algorithm: Two Point Crossover

*parent*1	0111 ↕ 1011 ↕ 0010	⇒	*child*1	0111 **0011** 0010
*parent*2	**1100** ↕ **0011** ↕ **1001**	⇒	*child*2	**1100** 1011 **1001**

Two Point Crossover

Two point crossover swaps the data between the parent chromosomes at two predefined points. Table.5.3 shows an example of two point crossover.

Cut And Splice Crossover

Cut and splice approach can generate new generations of various length. At two different points in two parents, it swaps the data and generates children of different length. Table 5.4 shows an example of cut and splice crossover.

5.1.6 Mutation

Mutation is an operator that is used to maintain genetic diversity from one generation to the next generation. The mutation methods could be variously dependant on particular problem. Simple one is to just select some genes from a chromosome and replace them with a new value or a random value. Table 5.5 gives an example. The mechanism of mutation looks very simple, but it is vital to ensure genetic diversity within the population.

Table 5.4: Genetic Algorithm: Cut and Splice Crossover

*parent*1	0111 ↕ 10110010	⇒	*child*1	0111 **1001**
*parent*2	**11000011** ↕ **1001**	⇒	*child*2	**11000011** 10110010

Table 5.5: Genetic Algorithm: Mutation

parent	0111 **1** 0110 **0** 10	⇒	*child*	0111 **0** 0110 **1** 10

5.2 Genetic Algorithm Based NSP Mining

NSP mining has attracted increasing attentions in recent data mining research because it considers negative relationships between itemsets, which are ignored by PSP mining. However, the search space for mining negative patterns is much bigger than that for positive ones. When the support threshold is low, in particular, there will be huge amounts of negative candidates. This chapter proposes a Genetic Algorithm based algorithm, GA-NSP, to find NSP with novel crossover and mutation operations, which are efficient at passing good genes on to next generations.

The target problem has been described in Section 3.6.

Based on Genetic Algorithm, we obtain NSP by crossover and mutation; high frequent patterns are then selected to be parents to generate offspring. By going through many generations, it will obtain a new and relatively high-quality population.

Since the Genetic Algorithm based method cannot ensure locating all of NSP, a key issue of using Genetic Algorithm in NSP mining is how to find all NSP. We therefore use an incremental population, and add all negative patterns, which are generated by crossover and mutation during the evolution process, into population. A dynamic fitness function is proposed to control population evolution. Ultimately, we can secure almost all the frequent patterns. The proportion can capture more than 90% in our experiments on two synthetic datasets.

5.3 GA-NSP Algorithm

The general idea of the proposed algorithm is shown as Figure 5.1. We will describe it from how to encode a sequence, and then introduce what are

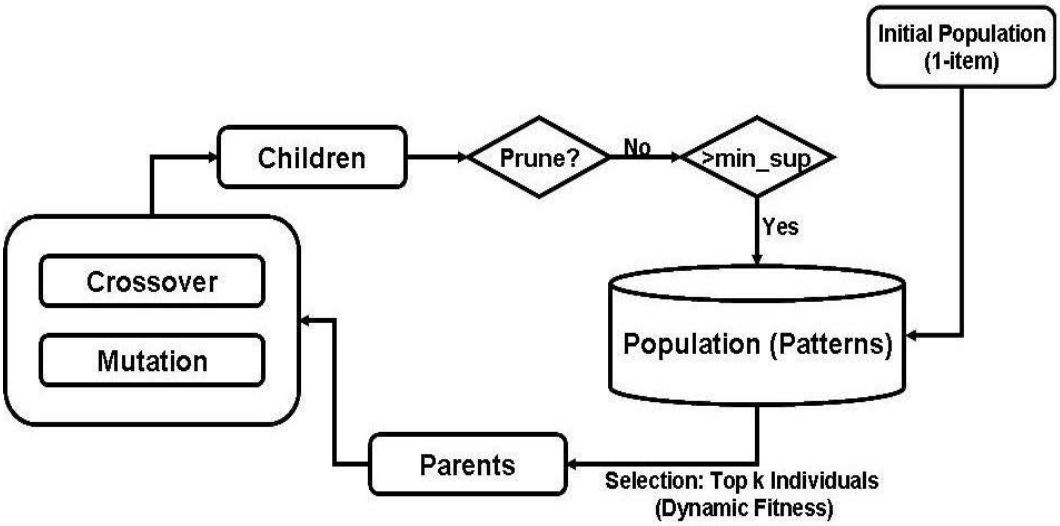


Figure 5.1: GA-NSP Algorithm: Process Flow

Table 5.6: GA-NSP Algorithm: Encoding

Sequence		Chromosome		
		$gene_1$	$gene_2$	$gene_3$
$<a\ b\ \neg(c,d)>$	$\Rightarrow$	$+a$	$+b$	$(\neg c \neg d)$

population, selection, crossover, mutation, pruning, fitness function and so on. A detailed algorithm will then be introduced.

5.3.1 Encoding

Sequence is mapped into a chromosome code in Genetic Algorithm. Both crossover and mutation operations depend on the chromosome code. We need to define the chromosome to represent the problems of NSP mining exactly. There are many different methods to encoding the chromosome, such as binary encoding, permutation encoding, value encoding and tree encoding (Mitchell 1996). The permutation encoding method is suitable for ordering problem and its format is consistent with the format of the sequence data, so we use it for sequence encoding.

Each sequence is mapped into a chromosome. Each element of the sequence is mapped into a gene in the chromosome, no matter whether the element has one item or more. Given a sequence $<e_1\ e_2\ ...\ e_n>$, it is transformed to a chromosome which has n-genes. Each gene is composed of a tag and an element. The element includes one or more items, and the tag indicates that the element is positive or negative. For example, a negative sequence $<a\ b\ (\neg c \neg d)>$ is mapped into a 3-gene chromosome, see Table 5.6.

5.3.2 Population

In the classical Genetic Algorithm method, the number of populations is fixed (Mitchell 1996). It uses a fixed number of populations to produce the next generation, but the populations tended to contract into one high frequent pattern, and it can only obtain a small part of frequent patterns. To achieve as many sequential patterns as possible, we potentially needed a population to cover more individuals. We therefore adjusted the basic Genetic Algorithm to suit NSP mining in the following ways.

Initial Population

All 1-item frequent positive patterns are obtained first. Based on the 1-item positive patterns, we transform all of them to their corresponding 1-item negative sequences, such as transforming the frequent positive sequence $<e>$ to the negative sequence $<\neg e>$. We then take all positive and negative 1-item patterns as the initial population.

Population Increasing

We do not limit population to a fixed number. When we acquire new sequential patterns during the evolvement, new patterns are put into the population for the next selection. If the population has already included the patterns, we ignore them. To improve the performance of this process, a hash table is used to verify whether a pattern has already appeared in the population.

Table 5.7: GA-NSP Algorithm: Crossover

$parent1$	$b\ \neg c \updownarrow a$	$\Rightarrow$	$child1$	$b\ \neg c\ \boldsymbol{e}$
$parent2$	$\boldsymbol{d} \updownarrow \boldsymbol{e}$	$\Rightarrow$	$child2$	$\boldsymbol{d}\ a$

5.3.3 Selection

The commonly used selection method is roulette wheel selection (Haupt & Haupt 1998). We have an increased population and the population number depends on the count of sequential patterns; thus, we can not use roulette wheel selection because the selection will be too costly if the population number is huge. We select the top K individuals with high dynamic fitness (see Section 5.3.7), where K is a constant number showing how many individuals will be selected for the next generation. To improve the performance of this selection method, we sort all individuals in population in descending order by dynamic fitness value. In every generation, we only select the first K individuals.

5.3.4 Crossover

Parents with different lengths are allowed to crossover with each other, and crossover may happen at different positions to get sequential patterns with varied lengths. For example, a crossover takes place at a different position, which is shown by '$\updownarrow$' in Table 5.7. After crossover, it may acquire two children. $Child1$ $<b\ \neg c\ e>$ consists of the first part of $parent1$ and the second part of $parent2$. $Child2$ $<d\ a>$ consists of the second part of $parent1$ and the first part of $parent2$. So we get two children with different lengths. If a crossover takes place both at the end/head of $parent1$ and at the head/end of $parent2$, as Table 5.8 shows, $child2$ will be empty. In that case, we shall set $child2$ by reverse. A *Crossover Rate* is also used to control the probability of cross over when parents generate their children.

Table 5.8: GA-NSP Algorithm: Crossover at Head/End

*parent*1	$b\ \neg c\ a\ \updownarrow$	$\Rightarrow$	*child*1	$b\ \neg c\ a\ \boldsymbol{d}\ \boldsymbol{e}$
*parent*2	$\updownarrow\ \boldsymbol{d}\ \boldsymbol{e}$	$\Rightarrow$	*child*2	$\boldsymbol{d}\ \boldsymbol{e}\ b\ \neg c\ a$

5.3.5 Mutation

Mutation is helpful in avoiding contraction of the population to a special frequent pattern. To introduce mutation into sequence generation, we select a random position and then replace all genes after that position with 1-item patterns. For example, given an individual $<b\ \neg c\ a>$, after mutation, it may change to $<b\ d\ \neg e>$ if $<d>$ and $<\neg e>$ are 1-item patterns. *Mutation Rate* is a percentage to indicate the probability of mutation when parents generate their children.

5.3.6 Pruning

When a new generation is obtained after crossover and mutation, it is necessary to verify whether the new generation is valid in terms of the constraints for NSP before passing over the whole dataset for their supports.

For a new individual $c=<e_1\ e_2\ e_3\ ...\ e_n>$, $c'=<e_i\ e_j\ ...\ e_k>$ $(0<i\leq j\leq k\leq n)$ is the max positive subsequence of c, that is to say, e_i, e_j, ... and e_k are all positive elements, and other elements in c are negative. If c' is not frequent, c must be infrequent and should be pruned. This method is simple but effective for pruning invalid candidates without cutting off possible valid individuals by mistake.

5.3.7 Fitness Function

Fitness

In order to evaluate the individuals and decide which are the best for the next generation, a fitness function for individuals is implemented in Genetic

Algorithm. We use the fitness function shown in Equation.5.1:

$$ind.fitness = (ind.support - min\_sup) \times DatasetSize. \tag{5.1}$$

The fitness function is composed of two parts. *Support* is the percentage that indicates how many proportion records are matched by the individual. If support is high, fitness will be high, so that the individual has good characteristics to pass down to next generation. *min_sup* is a threshold percentage value for verifying whether a sequence is frequent. *Dataset size* is the record count of whole sequence database.

Dynamic Fitness

Because the characteristics of the individual have been transmitted to the next generation by crossover or mutation, the individual should exit after a few generations. The result will tend to contract to one point if the individual doesn't exit gradually. We therefore set a dynamic fitness $dfitness$ to every individual in the population, shown in Equation.5.2. Its initial value is equal to $fitness$, but decreases during the evolvement. It indicates that the individuals in the population will gradually ceased to evolve. It is like a life value. When an individual's dynamic fitness is low or close to 0(<0.01), we set it to 0 because we regard it as a wasted individual which cannot be selected for the next generation.

$$ind.dfitness = \begin{cases} ind.fitness, & \text{initial set} \\ ind.dfitness \times (1 - DecayRate), & \text{if } ind \text{ is selected} \end{cases} \tag{5.2}$$

Decay Rate. We set a decay rate to indicate the decrease speed of individual's fitness. The decay rate is a percentage value between 0% and 100%. If an individual is selected by the selection process, its dynamic fitness will decrease by the speed of the decay rate. If the decay rate is high, dynamic fitness will decrease quickly and individuals will quickly cease to evolve. Thus, we may get less frequent patterns through a high decay rate. If we

want to obtain the maximum frequent patterns, we can set a low decay rate, such as 5%, but this will give rise to a longer execution time.

How Fitness Function Guide Effective Search

The fitness function focuses on the support of an individual/pattern. Generally speaking, an individual would have high fitness if it is highly frequent. That makes higher frequent patterns get higher probability to be selected for next generation. That is why some patterns can evolve whereas others can not.

And the second key fitness function, dynamic fitness function, is to avoid convergence of some special highly frequent patterns. If a pattern is selected once, its dynamic fitness will be reduced by a predefined value, and that makes its dynamic fitness value decrease gradually until it is dead, which means it won't evolve anymore.

5.3.8 Algorithm Description

Our algorithm is composed of the following six steps.

Step 1: We obtain the initial population which includes all frequent 1-item positive and 1-item negative sequences.

Step 2: Calculate all initial individuals' fitness. Their *dynamic* $fitness$ is set to their $fitness$.

Step 3: We select the top K individuals with high dynamic fitness from the population. After selection, the dynamic fitness of the selected individuals is updated by Equation.5.2.

Step 4: Crossover and mutation between the selected individuals to produce the next generation.

Step 5: After obtaining the next generation, we first prune invalid individuals and then calculate the frequency and fitness of remained individuals in new generation. If the frequency of an individual is greater than *min_sup*, we add it into the population, and set its fitness and dynamic fitness, but if the population has included this individual, we ignore it.

Step 6: Go back to step 3 and iterate the above process until all individuals in the population are dead (i.e., their dynamic fitness has become close to 0). The dead individuals are still in population, but they ceased to evolve. In the end, we obtain the final result - whole population, which is composed of all dead individuals.

The pseudocode of our algorithm is given as follows.

GA-NSP Mining Algorithm

Function: Find negative sequential patterns from a sequence database.

Input: Sequence database D, frequency threshold min_sup, decay rate threshold $decay\_rate$, crossover rate threshold $Crossover\_Rate$, mutation rate threshold $Mutation\_Rate$.

Output: NSP set pop, which contains the best individuals.

```
RunGA(min_sup, decay_rate, crossover_rate, mutation_rate){
01.     pop = initialPopulation();
02.     for (each individual ind in pop){
03.         ind.fitness = calculateFitness(ind);
04.         ind.dfitness = ind.fitness
05.         pop.sum_dfitness = pop.sum_dfitness + ind.dfitness
06.     }
07.     while ( pop.sum_dfitness > 0 ){
08.         popK = Selection(pop);
09.         if (Random()<crossover_rate) Crossover(popK);
10.         if (Random()<mutation_rate) Mutation(popK);
11.         for (each individual ind in popK) {
12.             if (Prune(ind) !=true && ind.sup >= min_sup)
13.                 pop.add(ind);
```

```
14.          }
15.      }
16.       return pop;
}
```

Selection(*pop*){ ▷ Subfunction for selecting top K individuals from population

```
01.      for (each ind with top K dfitness in pop){
02.          popK.add(ind);
03.          ind.dfitness = ind.dfitness * (1-decay_rate);
04.          if (ind.dfitness < 0.01)
05.              ind.dfitness = 0;
06.          }
07.      return popK;
}
```

5.3.9 An Example of GA-NSP Algorithm

It may not be easy to understand the whole process of GA-NSP algorithm. A simple example is listed here to make it clearer.

The example comes from one of the experiments in Section 5.4. It uses Dataset 1 (DS1) as data source, and *min_sup* is set at 0.24.

At first, it uses a PSP mining algorithm, such as GSP, to get frequent 1-item patterns, which is used as the initial population. In the example, the initial population is as following. *count* means the support count of the pattern, and *dfitness* is the dynamic fitness value of the pattern.

Initial-Population[0]= <91> , count=2578.0, dfitness=464.08
Initial-Population[1]= <414> , count=2325.0, dfitness=211.08
Initial-Population[2]= <440> , count=2266.0, dfitness=152.08
Initial-Population[3]= <538> , count=2209.0, dfitness=95.08

Initial-Population[4]= <646> , count=2707.0, dfitness=593.08
Initial-Population[5]= <663> , count=2375.0, dfitness=261.08

And then, all corresponding negative 1-item patterns are added into the initial population as well. Their *count* and $dfitness$ are set to zero.

Initial-Population[6]= <-91> , count=0.0, fitness=0.01
Initial-Population[7]= <-414> , count=0.0, fitness=0.01
Initial-Population[8]= <-440> , count=0.0, fitness=0.01
Initial-Population[9]= <-538> , count=0.0, fitness=0.01
Initial-Population[10]= <-646> , count=0.0, fitness=0.01
Initial-Population[11]= <-663> , count=0.0, fitness=0.01

At this step, an initial population is formed and all these individuals in the initial population will be put into final output patterns set *patterns*. In the *patterns*, all individuals are sorted by $dfitness$ to make following selection operation run quickly.

Now evolution process starts. In the first generation, a population are selected from the final patterns set *patterns*. We set the max count of population at 20. Usually, it choose top 20 individuals with high $dfitness$. Since there are only 12 individuals in the *patterns* in the example, all of them are selected.

---

pop-individuals[0]= <-663>
pop-individuals[1]= <663>
pop-individuals[2]= <-646>
pop-individuals[3]= <646>
pop-individuals[4]= <-538>
pop-individuals[5]= <538>
pop-individuals[6]= <-440>
pop-individuals[7]= <440>

pop-individuals[8]= <-414>
pop-individuals[9]= <414>
pop-individuals[10]=<-91>
pop-individuals[11]=<91>

After the selection, it runs crossover and mutation on *pop-individuals*, as following:

---

begin crossover:
father=<-663>
mother=<663>
–>>child 1: <663>
–>>child 2: <-663>
father=<-646>
mother=<646>
–>>child 1: <-646>
–>>child 2: <646>
father=<-538>
mother=<538>
–>>child 1: <-538>
–>>child 2: <538>
father=<-440>
mother=<440>
–>>child 1: <-440 440>
–>>child 2: <440 -440>
father=<-414>
mother=<414>
–>>child 1: <-414>
–>>child 2: <414>
father=<-91>
mother=<91>

–>>child 1: <-91 91>

–>>child 2: <91 -91>

begin mutation:

–>> null

In the previous steps, parents <440>, <-440>, <91> and <-91> can generate 2-item children since the crossover happens at both head and end, see Table 5.8 for details. No mutation happens up to now.

Now it gets a few new generations, such as <-440 440>, <440 -440>, <-91 91> and <91 -91>. Their support count and dynamic fitness values are calculated. If any of them has greater support than min_sup, then it is added into *patterns* and sorted by $dfitness$. The *patterns* would contain the following values at this moment.

---

patterns[0]= <646> , count=2707.0, dfitness=563.43

patterns[1]= <91 -91> , count=2578.0, dfitness=464.08

patterns[2]= <91> , count=2578.0, dfitness=440.88

patterns[3]= <663> , count=2375.0, dfitness=248.03

patterns[4]= <414> , count=2325.0, dfitness=200.53

patterns[5]= <440 -440> , count=2266.0, dfitness=152.08

patterns[6]= <440> , count=2266.0, dfitness=144.48

patterns[7]= <538> , count=2209.0, dfitness=90.33

patterns[8]= <-663> , count=0.0, dfitness=0.0

patterns[9]= <-538> , count=0.0, dfitness=0.0

patterns[10]= <-414> , count=0.0, dfitness=0.0

patterns[11]= <-91> , count=0.0, dfitness=0.0

patterns[12]= <-440> , count=0.0, dfitness=0.0

patterns[13]= <-646> , count=0.0, dfitness=0.0

The $dfitness$ values are updated since the corresponding patterns are selected once. For example, <646> was selected in the previous steps. Its

$dfitness$ value was 593.08, now its $dfitness$= 593.08*(1-$decay\_rate$) = 593.08 * (1-0.05) = 563.43, as the above $patterns$[0].

Then it starts next generation from selection, crossover and mutation again as the same way, until the $dfitness$ of all individuals in $patterns$ are close to zero.

Hereby we just list the No.200 generation results to see how it evolves.

---

patterns[0]= <-538 646> , count=2233.0, dfitness=49.79
patterns[1]= <-538 646 -646> , count=2178.0, dfitness=29.69
patterns[2]= <646 -646> , count=2707.0, dfitness=17.22
patterns[3]= <646 -663> , count=2336.0, dfitness=7.14
patterns[4]= <646 -440> , count=2390.0, dfitness=3.02
patterns[5]= <91 -646> , count=2213.0, dfitness=2.60
patterns[6]= <414 -414> , count=2325.0, dfitness=2.09
patterns[7]= <646 -414> , count=2339.0, dfitness=2.01
patterns[8]= <91 -663> , count=2217.0, dfitness=1.99
patterns[9]= <91 -538> , count=2284.0, dfitness=1.86
patterns[10]= <91 -414> , count=2254.0, dfitness=1.62
patterns[11]= <646 -538> , count=2406.0, dfitness=1.56
patterns[12]= <91 -440> , count=2269.0, dfitness=1.53
patterns[13]= <646 -91> , count=2315.0, dfitness=0.0
patterns[14]= <646> , count=2707.0, dfitness=0.0
patterns[15]= <-663> , count=0.0, dfitness=0.0
patterns[16]= <-91> , count=0.0, dfitness=0.0
patterns[17]= <-538> , count=0.0, dfitness=0.0
patterns[18]= <91 -91> , count=2578.0, dfitness=0.0
patterns[19]= <-646> , count=0.0, dfitness=0.0
patterns[20]= <538> , count=2209.0, dfitness=0.0
patterns[21]= <538 -538> , count=2209.0, dfitness=0.0
patterns[22]= <-91 646> , count=2146.0, dfitness=0.0
patterns[23]= <440 -440> , count=2266.0, dfitness=0.0

patterns[24]= <91> , count=2578.0, dfitness=0.0
patterns[25]= <440> , count=2266.0, dfitness=0.0
patterns[26]= <663> , count=2375.0, dfitness=0.0
patterns[27]= <-440> , count=0.0, dfitness=0.0
patterns[28]= <-414> , count=0.0, dfitness=0.0
patterns[29]= <414> , count=2325.0, dfitness=0.0

5.4 Experiments

The proposed algorithm is implemented with Java and tested with three synthetic sequence datasets generated by an IBM data generator (Agrawal & Srikant 1995) and a real-world dataset. We also implemented the PNSP algorithm (Hsueh et al. 2008) and Neg-GSP algorithm (Zheng et al. 2009) with Java for performance comparison. All the experiments were conducted on a PC with Intel Core 2 CPU of 2.9GHz, 2GB memory and Windows XP Professional SP2.

Dataset 1 (DS1) is C8.T8.S4.I8.DB10k.N1k, which means the average number of elements in a sequence is 8, the average number of items in an element is 8, the average length of a maximal pattern consists of 4 elements and each element is composed of 8 items average. The data set contains 10k sequences, the number of items is 1000. The minimum number of elements in a sequence is 1, and the maximum number is 237.

Dataset 2 (DS2) is C10.T2.5.S4.I2.5.DB100k.N10k, which means the average number of elements in a sequence is 10, the average number of items in an element is 2.5, the average length of a maximal pattern consists of 4 elements and each element is composed of 2.5 items average. The data set contains 100k sequences, the number of items is 10k. The minimum number of elements in a sequence is 1, and the maximum number is 138.

Dataset 3 (DS3) is C20.T4.S6.I8.DB10k.N2k, which means the average number of elements in a sequence is 20, the average number of items in an

Table 5.9: GA-NSP Algorithm: Features of Synthetic Datasets

Parameters	*DS1*	*DS2*	*DS3*
Number of sequences (DB)	10k	100k	10k
Number of items (N)	1k	10k	2k
Average number of elements per sequence (C)	8	10	20
Average number of items per element (T)	8	2.5	4
Average length of maximal potentially large sequences (S)	4	4	6
Average size of itemsets in maximal potentially large sequences (I)	8	2.5	8

element is 4, the average length of a maximal pattern consists of 6 elements and each element is composed of 8 items average. The data set contains 10k sequences, the number of items is 2k. The minimum number of elements in a sequence is 1, and the maximum number is 301.

Dataset 4 (DS4) is real application data for insurance claims. The data set contains 479 sequences. The average number of elements in a sequence is 30. The minimum number of elements in a sequence is 1, and the maximum number is 171.

Features of the above three synthetic datasets are listed in Table 5.9.

Experiments were done to compare the different *Crossover Rate*, *Mutation Rate* and *Decay Rate* on two synthetic datasets, $DS1$ and $DS2$. Each experiment was run 10 times and then the average value was got as the final result. We focused on comparing execution time, the number of patterns and the execution time per pattern, which indicates how long it takes to get one pattern. The total number of all patterns was determined by PNSP and Neg-GSP algorithm, and it was then easy to know the proportion of patterns we could get by using our algorithm. The Y axis (see the 2nd charts of Figure 5.2 and Figure 5.3) indicates the proportion of patterns.

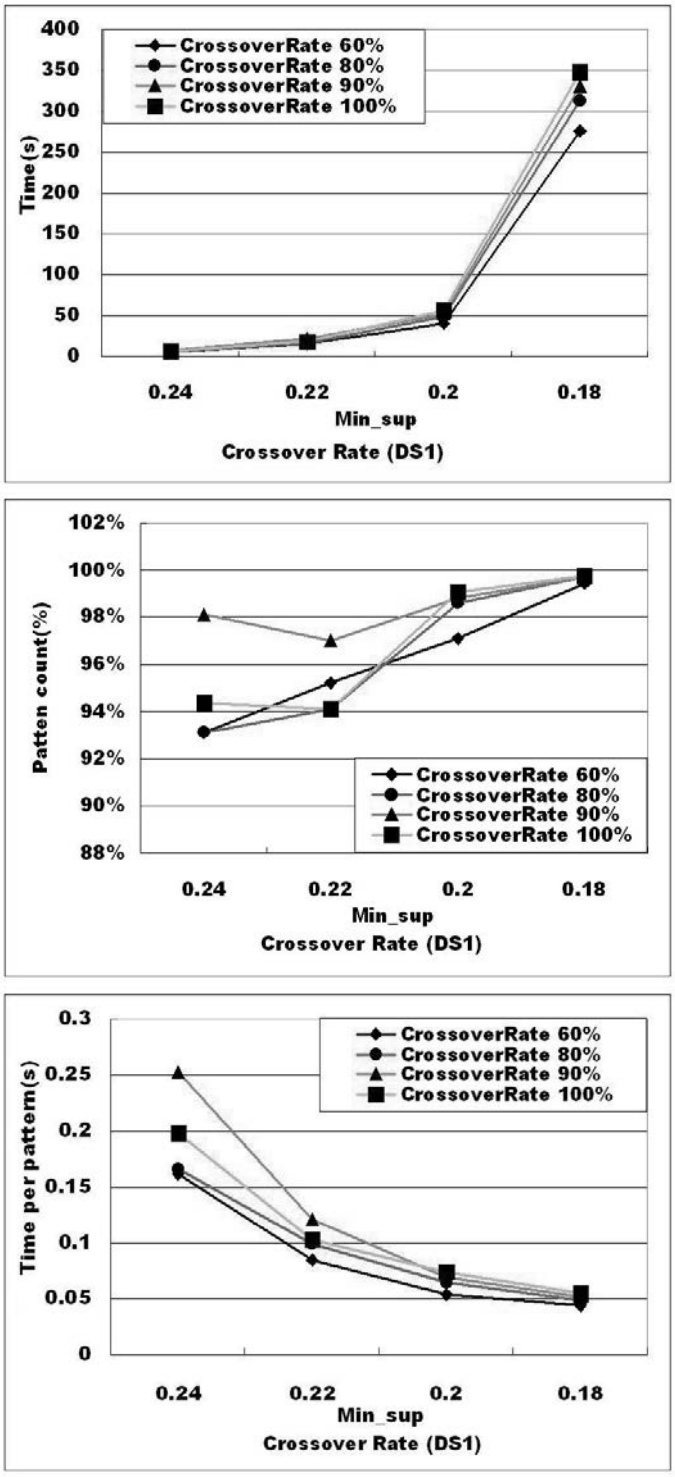


Figure 5.2: Experiments: Different Crossover Rates On DS1

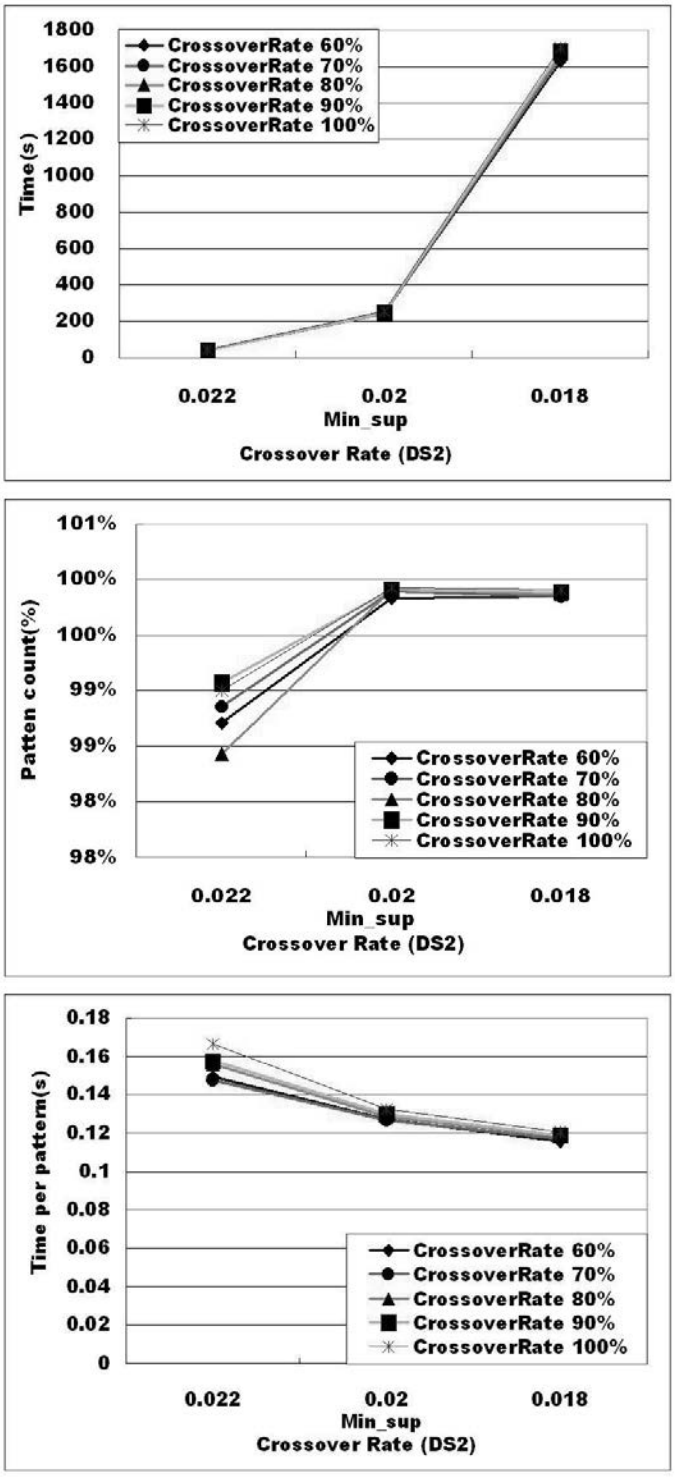


Figure 5.3: Experiments: Different Crossover Rates On DS2

5.4.1 Analysis of Crossover Rate

We compared different crossover rates from 60% to 100%. Figure 5.2 and 5.3 show the effect of different crossover rates on $DS1$ and $DS2$. With low crossover rates, such as 60%, we obtained almost the same proportion of patterns as with high crossover rates (see the 2nd charts in Figure 5.2 and Figure 5.3). The least execution-time per pattern is achieved when the crossover rate is low, so 60% is the best choice for the two datasets in our experiments.

5.4.2 Analysis of Mutation Rate

We compared different mutation rates from 0% to 20% on $DS1$ and $DS2$ (see Figure 5.4 and Figure 5.5). They show that the mutation rate will not have an outstanding effect, but if it is set to 0%, it will result in missing a lot of patterns. A Mutation rate of 5-10% is a good choice because it can produce around 80% patterns for $DS1$ and above 90% patterns for $DS2$. When the mutation rate is 5%, the average execution-time per pattern is lower. We therefore set a mutation rate of 5% for the following experiments.

5.4.3 Analysis of Decay Rate

Decay rate is a variable that we used to control evolution speed. If the decay rate is high, individuals will die quickly, so we can get only small proportion of patterns. If the decay rate is low, we can get more patterns, but a longer execution time is necessary (see Figure 5.6 and Figure 5.7). In order to get all NSP, we always choose *decay rate*=5%, which enables us to obtain around 90% to 100% patterns on the two datasets.

5.4.4 Performance Comparison with Other Methods

We compared our algorithm with PNSP and Neg-GSP, which are two algorithms proposed recently for NSP mining. The tests are based on $Crossover$

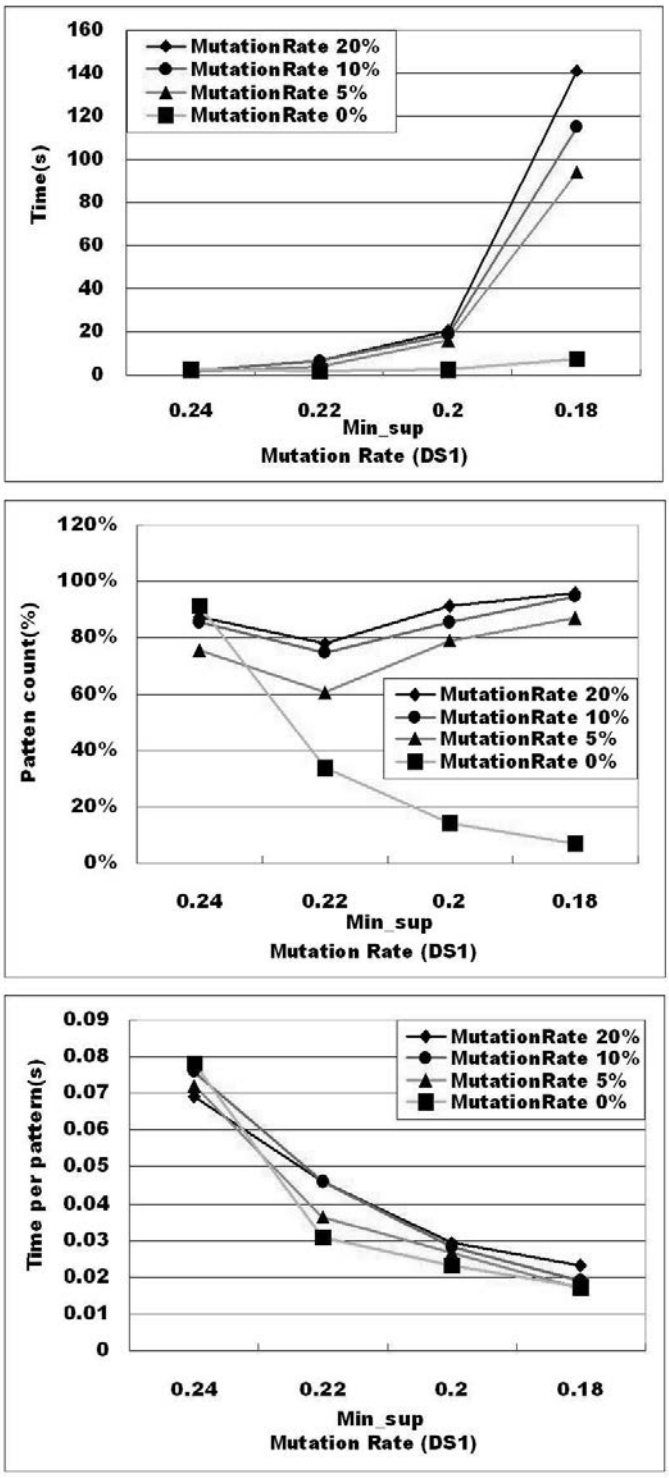


Figure 5.4: Experiments: Different Mutation Rates On DS1

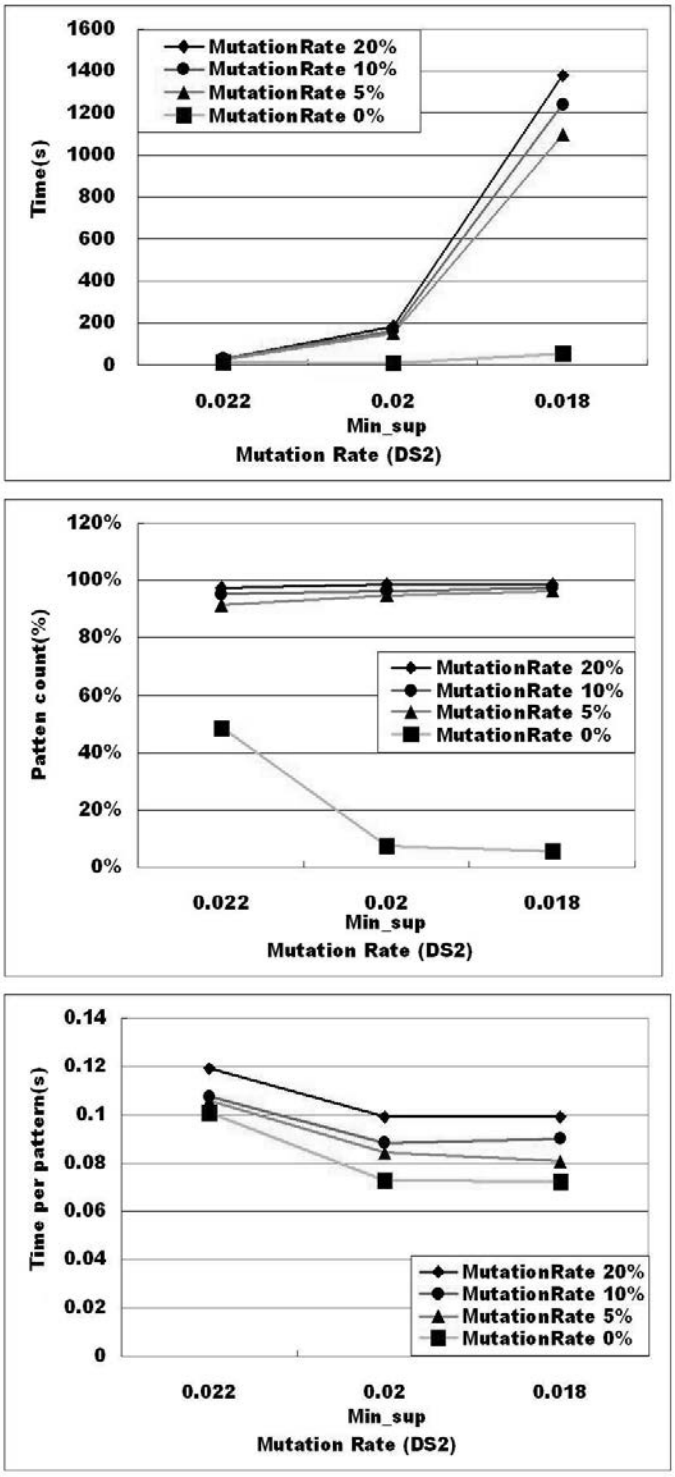


Figure 5.5: Experiments: Different Mutation Rates On DS2

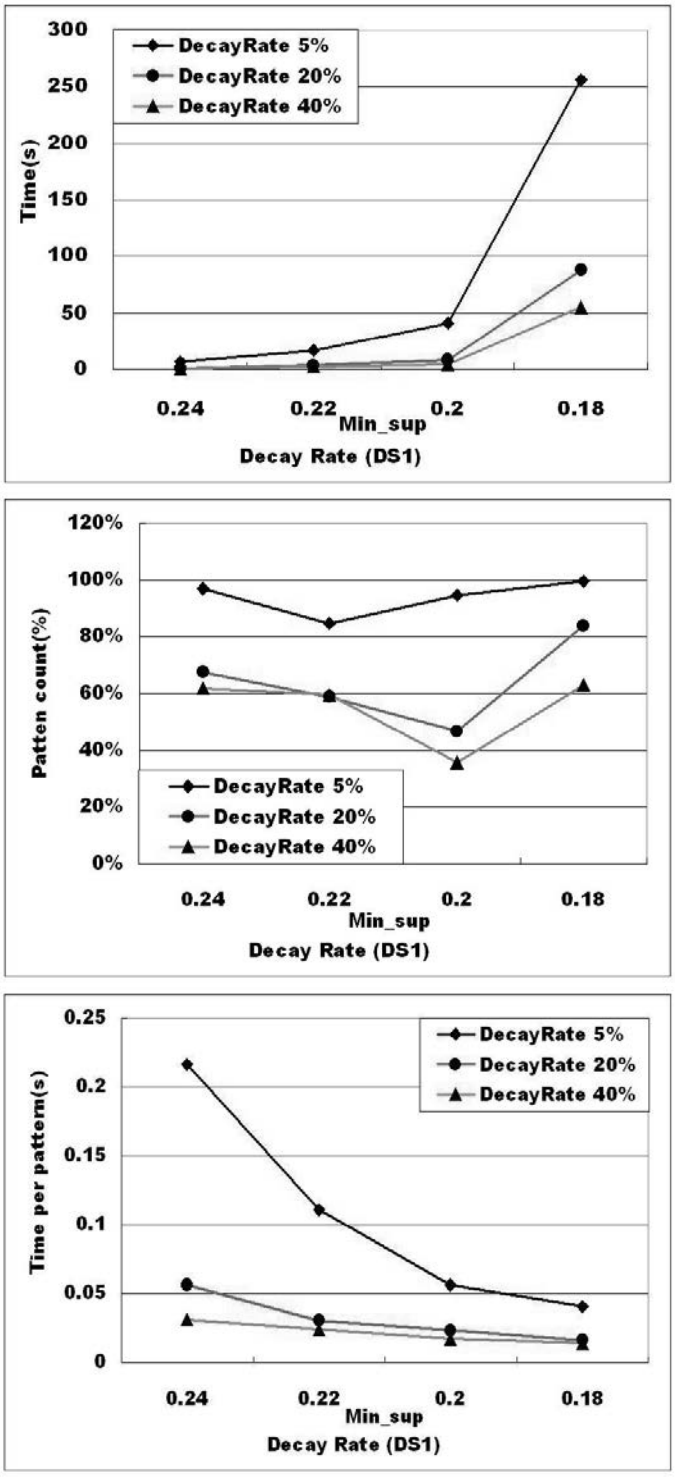


Figure 5.6: Experiments: Different Decay Rates On DS1

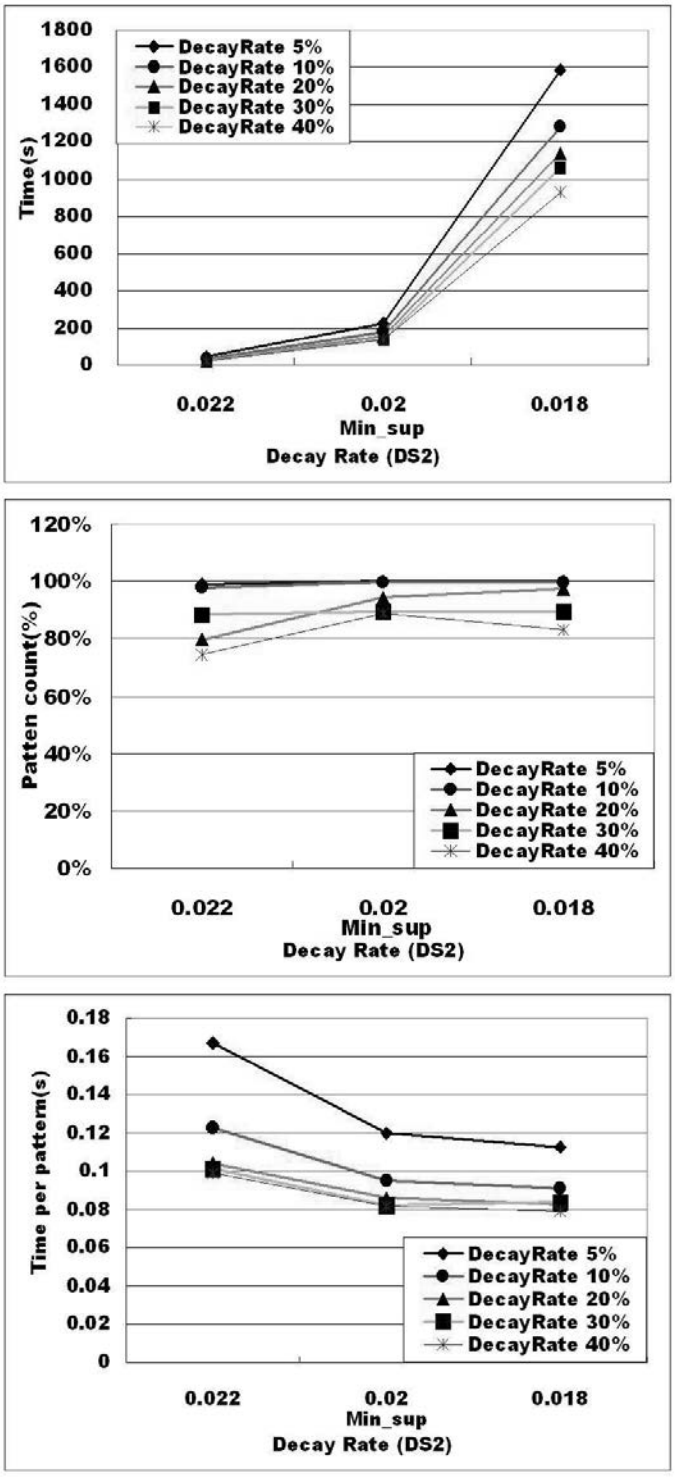


Figure 5.7: Experiments: Different Decay Rates On DS2

Rate=60%, *Mutation Rate*=5% and *Decay Rate*=5%. The results (see Figure 5.8 to 5.11) on 4 different datasets show that the performance of the GA-NSP algorithm is better than PNSP and Neg-GSP when the support threshold is low. Our algorithm is not better than others when *min_sup* is high, because most patterns are very short and the GA-NSP method cannot demonstrate its advantage.

When *min_sup* is high, there are not as many patterns and the patterns are short, so it is very easy to find the patterns with existing methods. However, when *min_sup* is low, the patterns are longer and the search space is much bigger, so it is time-consuming to find patterns using traditional methods. Using our GA-NSP algorithm, it is still can obtain the patterns quickly even though *min_sup* is very low.

5.5 Conclusions

This chapter proposes a Genetic Algorithm based algorithm, GA-NSP, to find NSP with novel crossover and mutation operations. An effective dynamic fitness function and a pruning method are also provided to improve performance. The results of extensive experiments show that the proposed method can find negative patterns efficiently and has remarkable performance compared with some other algorithms of negative pattern mining. It specially outperforms existing algorithms when the support threshold *min_sup* is low or when the patterns are long.

- Genetic Algorithm are efficient at passing good genes on to next generations in NSP mining;
- The fitness function focuses on the support of an individual / pattern. A highly frequent individual would have high fitness. That makes the highly frequent patterns have higher probability to be selected for next generation. That is why some patterns can evolve whereas others can not.

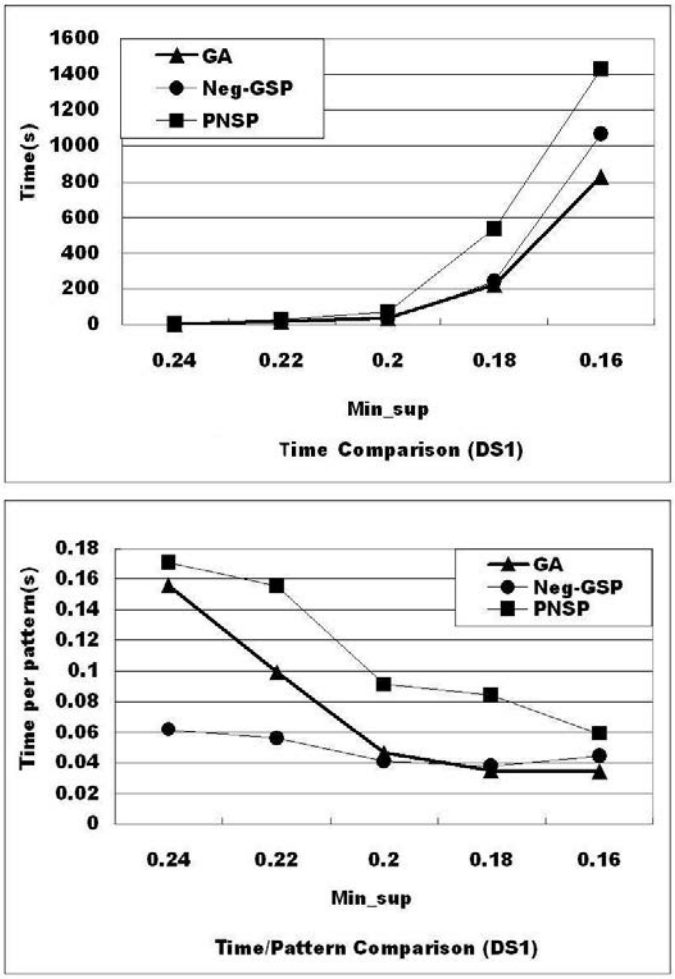


Figure 5.8: GA-NSP Algorithm: Execution Time Comparison On DS1

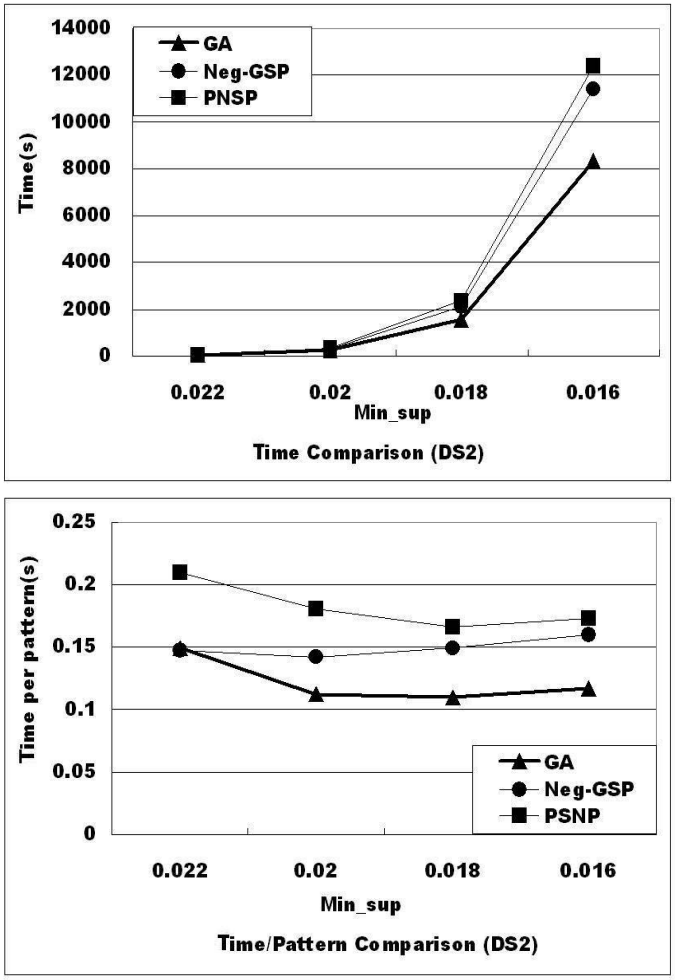


Figure 5.9: GA-NSP Algorithm: Execution Time Comparison On DS2

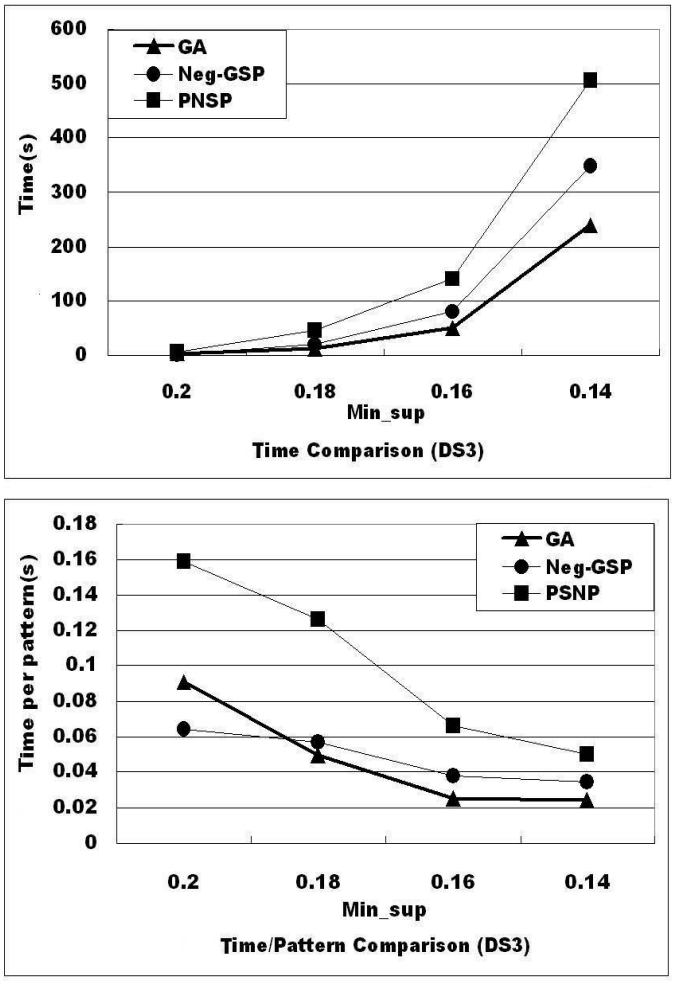


Figure 5.10: GA-NSP Algorithm: Execution Time Comparison On DS3

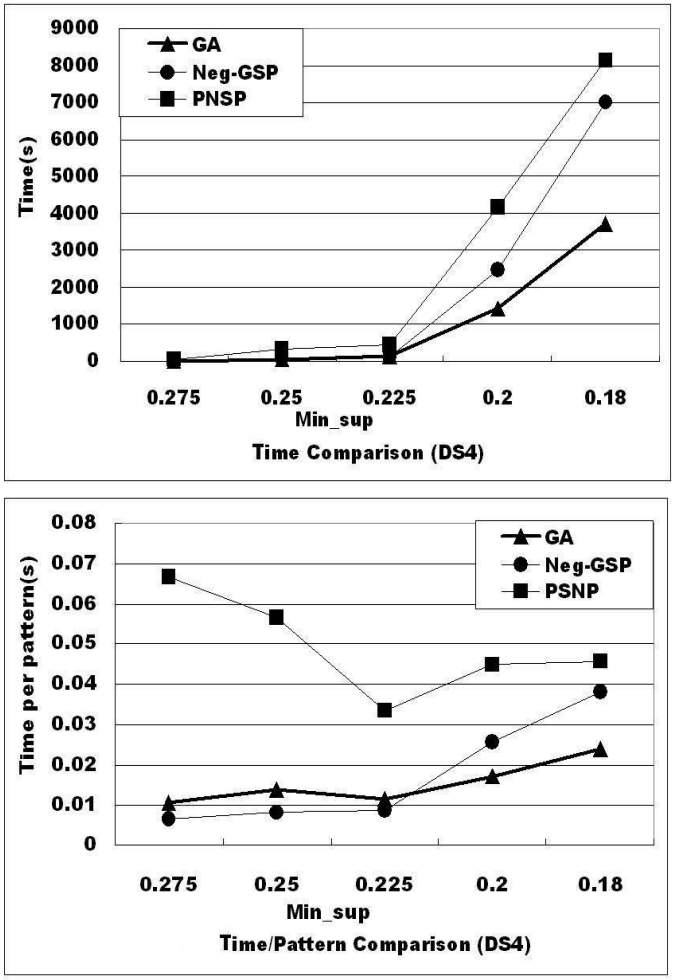


Figure 5.11: GA-NSP Algorithm: Execution Time Comparison On DS4

- Dynamic fitness function can avoid convergence of some special highly frequent patterns. If a pattern is selected once, its dynamic fitness will be reduced by a predefined value, and that makes its dynamic fitness value decrease gradually until it is dead, which means it won't evolve anymore.

Chapter 6

Effective NSP (e-NSP) Mining Algorithm

The typical approach of getting support of NSC is to pass over the whole sequence database, and Neg-GSP algorithm in Chapter 4 is one of them. But it is obviously not efficient enough. We try to find a method that can avoid passing over the sequence database many times so as to reduce execution time of calculating candidate's support. A feasible method is to calculate support of negative candidate only using the support info of corresponding positive patterns, without re-scanning database.

We focus on these area and got much progress so far. (Dong et al. 2011) proposed a deducible method, which is called e-NSP, to calculate support of negative candidate based on the support of identified positive patterns.

The idea of the proposed method comes from the Set Theory, it is assumed that there is special relationship between NSC and its corresponding PSP. Therefore, the support of a NSC can be calculated by using only the support information of the corresponding PSP, without any additional database scanning. The basic working mechanism of e-NSP is as follows.

First, negative containment is defined to determine whether or not a data sequence contains a negative sequence. It clearly defines the boundary of a data sequence set containing a NSC, and supports applying the Set Theory

in NSP mining.

Second, an efficient approach is proposed to convert the negative containing problem to a positive containing problem. The supports of NSC are then calculated based only on the corresponding PSP. In this way, mining NSP does not need additional database scans, and the existing PSP mining algorithms can be used to mine for NSP.

Finally, a simple but efficient approach is proposed to generate NSC. We test and compare our approach with two currently available NSP mining algorithms on both synthetic and real-life datasets. This clearly shows that e-NSP is much more efficient than any other available NSP approach. It also shows that e-NSP offers a new strategy for efficient mining of NSP in large datasets.

So far, we got only outcome for some special NSP with strong constraints, not for all general NSP, as described in the following problem statements.

6.1 Problem Statement

6.1.1 Related Definitions

Several concepts are defined in the following before we discuss e-NSP algorithm.

Definition 1. Positive Partner

The positive partner of a negative element $\neg e$ is e, denoted as $PP(\neg e)$, i.e., $PP(\neg e) = e$. The positive partner of positive element e is e itself, i.e., $PP(e) = e$. The positive partner of a negative sequence ns=$<s_1 \ldots s_k>$ is to change all negative elements in ns to their positive partners, denoted as $PP(ns)$, i.e., $PP(ns) = \{< s'_1 \ \ldots s'_k > |\ s'_i = PP(s_i), s_i \in ns\}$.

Example 1. *PP(<(¬a¬b) c ¬d>)=<(ab) c d>.*

Definition 2. Positive/Negative Element-id Set

Element id is the order number of an element in a sequence. Given a sequence $s =< s_1\ s_2\ \ldots s_m >$, $id(s_i) = i$ is the element id 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., $EidS_s = \{(s_i, id(s_i)) \mid s_i \in s\} = \{(s_1, 1), (s_2, 2), \ldots, (s_m, m)\}$ $(1 \leqslant i \leqslant m)$. The set including all positive and negative element-ids of a sequence s is called positive and negative element-id set of s, denoted as $EidS_s^+$, $EidS_s^-$ respectively.

Example 2. Given $s =< \neg(ab)\ c\ \neg d >$, $EidS_s$=*{(¬(ab),1), (c,2), (¬d,3)}*, $EidS_s^+ = \{(c, 2)\}$, $EidS_s^- = \{(\neg(ab), 1), (\neg d, 3)\}$.

Definition 3. Subsequence (Definition By Element-id Set)

Given a sequence s, $EidS'_s = \{(\alpha_1, id_1), (\alpha_2, id_2), \ldots, (\alpha_p, id_p)\}$ $(1 < p \leqslant m)$ is a subset of $EidS_s$, $\alpha =< \alpha_1\ \alpha_2 \ldots \alpha_p >$, if $\forall \alpha_i$, $\alpha_{i+1} \in \alpha$ $(1 \leqslant i < p)$, we have $id_i < id_{i+1}$, then α is called a subsequence of s, denoted as $\alpha = Sub(EidS'_s)$.

Example 3. Following Example 2, $s =< \neg(ab)\ c\ \neg d >$, $EidS_s = \{(\neg(ab), 1), (c, 2), (\neg d, 3)\}$, $EidS_s^+ = \{(c, 2)\}$, $EidS_s^- = \{(\neg(ab), 1), (\neg d, 3)\}$. $Sub(EidS_s^+) =< c >$. If $EidS'_s = \{(\neg(ab), 1), (c, 2)\}$, we can create a subsequence of s, $Sub(EidS'_s) =< \neg(ab)\ c >$.

Maximum Positive Subsequence and 1-neg-size Maximum Subsequence are two types of special subsequence; see the following definitions.

Definition 4. Maximum Positive Subsequence (Definition By Element-id Set)

Maximum positive subsequence is a special subsequence which is composed of all positive elements. Let $ns =< s_1\ s_2\ \ldots\ s_m >$ be an m-size and n-neg-size negative sequence $(m - n > 0)$, $Sub(EidS_{ns}^+)$ is called the maximum positive subsequence of ns, denoted as $MPS(ns)$. $MPS(ns) = Sub(EidS_{ns}^+)$.

Example 4. Given a negative sequence s=$<\neg(ab)\ c\ d >$, $EidS^+ =$

$\{(c,2),(d,3)\}$, its maximum positive subsequence is $MPS(s) =< c\ d >$.

Definition 5. 1-neg-size Maximum Subsequence

For a negative sequence ns, its subsequence that includes $MPS(ns)$ and one negative element e is called a 1-neg-size maximum subsequence, denoted as 1-negMS, 1-negMS=$Sub(EidS^+_{ns}, e)$, where $e \in EidS^-_{ns}$.

The subsequence set including all 1-neg-size maximum sub-sequences of ns is called 1-neg-size maximum subsequence set, denoted as 1-$negMSS_{ns}$, 1-$negMSS_{ns} = \{Sub(EidS^+_{ns}, e) \mid \forall e \in EidS^-_{ns}\}$. It is a subsequence which is composed of all positive elements and one negative element.

Example 5.1. $ns =< \neg(ab)\ c\ \neg d >$, 1-$negMSS_{ns} = \{< \neg(ab)\ c >, < c\ \neg(d) >\}$;

Example 5.2. $ns' =< \neg a\ (bc)\ d\ \neg(cde) >$, *1-negMSS*$_{ns'}$=$\{$*<¬a (bc) d>, <(bc) d ¬(cde)>*$\}$.

6.1.2 Constraints of Negative Candidates

In real applications, the number of NSP is huge, and many of them are not meaningful. In order to reduce the number of NSC and efficiently discover meaningful NSP, some constraints must be added to negative sequences (Hsueh et al. 2008)(Zheng et al. 2009). In this chapter, we only consider a negative sequence that satisfies the following three constraints.

Constraint 1. Frequency Constraint

In Section 3.2, the constraint 1 defined that: for each negative item in a NSP, its corresponding positive item is required to be frequent. For example, *(¬i)* is an interesting negative item if its positive item *(i)* is frequent. It just focuses on item. This chapter propose stronger constraint accordingly, which focuses on negative sequence, not only item. For a negative sequence ns, it must satisfy *sup(PP(ns))* $\geqslant$*min_sup*.

Example 6. Given frequent itemsets (ab) and d, we take their corresponding negative itemsets $\neg(ab)$ and $\neg d$ into account in a negative sequence, such as $< \neg(ab)\ c\ \neg d >$. However, if (ab) or d are not frequent, we ignore $\neg(ab)$ or $\neg d$ in our approach since we always focus on frequent positive itemsets and their corresponding negative itemsets.

Besides the frequency constraint, we defined another two types of format constraint as following. The two format constraints are described in Chapter 3, see section 3.2.

Constraint 2. Format Constraint 1

Adjacent negative elements in a NSC are not allowed.

Example 7. $< \neg(ab)\ c\ \neg d >$ satisfies Constraint 2, but $< \neg(ab)\ \neg c\ d >$ does not because of two adjacent negative elements $\neg(ab)$ and $\neg c$.

Constraint 2. Format Constraint 2

The minimum negative unit in a NSC is an element. If the element includes more than one item, either all items are positive or the whole element is negative. It is not permitted for certain items in the element to be negative while others are not negative (Zheng et al. 2009).

Example 8. $< \neg(ab)\ c\ d >$ satisfies Constraint 3, but $< (\neg ab)\ c\ d >$ does not because only $\neg a$ is negative in element $(\neg ab)$ while b is not.

In this chapter, we assume that a negative sequence implicitly satisfies the above three constraints.

6.1.3 Negative Containment

Negative containment defines how a data sequence contains a negative sequence/candidate. Before discussing the formal definition of negative containment, we introduce a related definition as following.

Definition 6. First Subsequence Ending Position / Last Subsequence Beginning Position

Given a data sequence $ds =< d_1\ d_2\ \ldots d_t >$ and a positive sequence α,

(1) if $\exists p(1 < p \leqslant t)$, $\alpha \subseteq < d_1 \ldots d_p > \wedge \alpha \not\subseteq < d_1 \ldots d_{p-1} >$, then p is called the First Subsequence Ending Position, denoted as $FSE(\alpha, ds)$; if $\alpha \subseteq < d_1 >$ then $FSE(\alpha, ds) = 1$;

(2) if $\exists q(1 \leqslant q < t)$, $\alpha \subseteq < d_q \ldots d_t > \wedge \alpha \not\subseteq < d_{q+1} \ldots d_t >$, then q is called the Last Subsequence Beginning Position, denoted as $LSB(\alpha, ds)$; if $\alpha \subseteq < d_t >$ then $LSB(\alpha, ds) = t$;

(3) if $\alpha \not\subseteq ds$, then $FSE(\alpha, ds) = 0$, $LSB(\alpha, ds) = 0$.

Example 9. Given $ds =< a\ (bc)\ d\ (cde) >$. $FSE(< a >, ds) = 1$, $FSE(< c >, ds) = 2$, $FSE(< c\ d >, ds) = 3$, $LSB(< a >, ds) = 1$, $LSB(< c >, ds) = 4$, $LSB(< c\ d >, ds) = 2$, $LSB(< (cd) >, ds) = 4$.

Our definition of a data sequence containing a negative sequence is as follows.

Definition 7. Negative Supporting Definition

Let $ds =< d_1\ d_2\ \ldots d_t >$ be a data sequence, $ns =< s_1\ s_2\ \ldots s_m >$ be an m-size and n-neg-size negative sequence, (1) if $m > 2t + 1$, then ds does not support ns; (2) if $m = 1$ and $n = 1$, then ds supports ns when $PP(ns) \not\subseteq ds$; (3) otherwise, ds supports ns if, $\forall(s_i, id(s_i)) \in EidS^-_{ns}(1 \leqslant i \leqslant m)$, any one of the following three conditions holds:

a) $(lsb = 1)$ or $(lsb > 1) \wedge PP(s_1) \not\subseteq < d_1 \ldots d_{lsb-1} >$, when $i = 1$,

b) $(fse = t)$ or $(0 < fse < t) \wedge PP(s_m) \not\subseteq < d_{fse+1} \ldots d_t >$, when $i = m$,

c) $(fse > 0 \wedge lsb = fse + 1)$ or $(fse > 0 \wedge lsb > fse{+}1) \wedge PP(s_i) \not\subseteq < d_{fse+1} \ldots d_{lsb-1} >$, when $1 < i < m$,

where $fse = FSE(MPS(< s_1\ s_2\ \ldots s_{i-1} >), ds)$, $lsb = LSB(MPS(< s_{i+1} \ldots s_m >), ds)$.

In the above definition, case a) indicates that the first element is negative in ns. "$(lsb > 1) \wedge PP(s_1) \not\subseteq < d_1 \ldots d_{lsb-1} >$" means that $< d_{lsb} \ldots d_t >$ contains $MPS(< s_2 \ldots s_m >)$ but $< d_1 \ldots d_{lsb-1} >$ does not contain $PP(s_1)$.

"$lsb = 1$" means that the last subsequence's beginning position is 1, so $PP(s_1)$ cannot be contained by ds. Case b) indicates that the last element is negative in ns. Case c) indicates that the negative element is between the first and last element in ns. "$lsb > fse + 1$" ensures there is at least one element in "$< d_{fse+1} \dots d_{lsb-1} >$". "$fse > 0 \wedge lsb = fse + 1$" means that d_{fse} and d_{lsb} are contiguous elements, so $PP(s_i)$ cannot be contained between them.

Example 10. Given $ds =< a\ (bc)\ d\ (cde) >$, we have

1) $ns =< \neg a\ c >$. $EidS^{-}_{ns} = \{(\neg a, 1)\}$. ds does not contain ns. $lsb = 4 > 0$, but $PP(s_1) =< a > \subseteq < d_1 \dots d_3 >=< a\ (bc)\ d >$ (Case a).

2) $ns =< \neg a\ a\ c >$. $EidS^{-}_{ns} = \{(\neg a, 1)\}$. ds contains ns because $lsb = 1$ (Case a).

3) $ns =< (ab)\ \neg(cd) >$. $EidS^{-}_{ns} = \{(\neg(cd), 2)\}$. ds does not contain ns because $fse = 0$ (Case b).

4) $ns =< (de)\ \neg(cd) >$. $EidS^{-}_{ns} = \{(\neg(cd), 2)\}$. ds contains ns because $fse = 4 (t = 4)$(Case b).

5) $ns =< a\ \neg d\ d\ \neg d >$. $EidS^{-}_{ns} = \{(\neg d, 2), (\neg d, 4)\}$. ds does not contain ns. For $(\neg d, 2), fse = 1, lsb = 4$, but $PP(\neg d) \subseteq < d_2 \dots d_3 >=< (bc)\ d >$ (Case c). If one negative element does not satisfy the condition, we do not need to consider other negative elements.

6) $ns =< a\ \neg b\ b\ \neg a\ (cde) >$. $EidS^{-}_{ns} = \{(\neg b, 2), (\neg a, 4)\}$. ds contains ns. For $(\neg b, 1)$, $fse = 1$, $lsb = 2$, $fse > 0 \wedge lsb = fse + 1$ (Case c); For $(\neg a, 4)$, $fse = 2$, $lsb = 4$, $PP(\neg a) \nsubseteq < d_3 >=< d >$ (Case c).

Negative sequences do not satisfy the Apriori property. The Apriori property can be simply described as: a sequence s is not frequent if any of its sub-sequences s' is not frequent, namely, $sup(s') \geqslant sup(s)$. However, as shown in Example 11, $sup(< \neg a\ c >) = 0$, $sup(<\neg a\ a\ c >) = 1$, $sup(< \neg a\ c >) < sup(< \neg a\ a\ c >)$, though $< a\ c > \subseteq < \neg a\ a\ c >$.

Given a data sequence $ds =< d_1\ d_2 \dots d_t >$, and $ns =< s_1\ s_2 \dots s_m >$, which is an *m-size* and *n-neg-size* negative sequence, the negative containment definition can be described as follows.

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-$negMS) \nsubseteq ds$.

Example 11. Given $ds =< a\ (bc)\ d\ (cde) >$, 1) if $ns =< a\ \neg d\ d\ \neg d >$, 1-$negMSS_{ns} = \{< a\ \neg d\ d >, < a\ d\ \neg d >\}$, then ds does not contain ns because $p(< a\ \neg d\ d >) =< a\ d\ d > \subseteq ds$; 2) if $ns' = < a\ \neg b\ b\ \neg a\ (cde) >$, 1-$negMSS'_{ns} = \{< a\ \neg b\ b\ (cde) >, < a\ b\ \neg a\ (cde) >\}$, then ds contains ns because $MPS(ns) =< a\ b\ (cde) > \subseteq ds \wedge p(< a\ \neg b\ b\ (cde) >) \nsubseteq ds \wedge p(< a\ b\ \neg a\ (cde) >) \nsubseteq ds$.

Based on the above definition, set properties are applicable in calculating $sup(ns)$.

6.1.4 Brief Introduction of Set Theory

Set theory is the branch of mathematics that studies sets, which are collections of objects.

A *set* is a collection of things (called the members or elements), the collection being regarded as a single object. For example, the set of prime numbers less than 14 is $\{2, 3, 4, 7, 11, 13\}$.

Union of the sets A and B, denoted $A \cup B$, is the set of all objects that are a member of A, or B, or both. For example, the union of $\{1, 2, 3\}$ and $\{2, 3, 4\}$ is the set $\{1, 2, 3, 4\}$.

Intersection of the sets A and B, denoted $A \cap B$, is the set of all objects that are members of both A and B. For example, the intersection of $\{1, 2, 3\}$ and $\{2, 3, 4\}$ is the set $\{2, 3\}$.

Set difference of U and A, denoted $U \setminus A$ is the set of all members of U that are not members of A. The set difference of $\{1, 2, 3\} \setminus \{2, 3, 4\}$ is $\{1\}$, while, conversely, the set difference of $\{2, 3, 4\} \setminus \{1, 2, 3\}$ is $\{4\}$. When A is a subset of U, the set difference $U \setminus A$ is also called the complement of A in U.

6.1.5 Negative Supporting

In the following, we discuss negative containment in set theory by giving an example. Figure 6.1 shows the intersection of sequence $< a >$ and $< b >$. $\{< a >\}$, $\{< b >\}$ mean the set of tuples that contain sequences $< a >$,$< b >$ in a sequence database respectively. There are three 2-length sequences: $< a\ b >$, $< b\ a >$ and $< (ab) >$, and four disjointed sets: $\{< (ab) >^{only}\}$, $\{< a\ b >^{only}\}$, $\{< b\ a >^{only}\}$ and $\{< a\ b >\} \cap \{< b\ a >\}$, which are the sets of tuples that contain sequences $< (ab) >$ only, $< a\ b >$ only, $< b\ a >$ only, and both $< a\ b >$ and $< b\ a >$ respectively.

Taking $\{< a\ \neg b >\}$ as an example, as seen in Figure 6.1, we have $\{< a\ \neg b >\} = \{< a > - < b >\} \cup \{< (ab) >^{only}\} \cup \{< b\ a >^{only}\}$

$= \{< a >\} - \{< a\ b >^{only}\} \cup (\{< ab >\} \cap \{< b\ a >\})$

$= \{< a >\} - \{< a\ b >\}$

This result is consistent with the negative containment definition, by which data sequences containing $\{< a\ \neg b >\}$ are the same sequences that contain $\{< a >\}$ but do not contain $\{< a\ b >\}$.

So we can get:

$sup(< a\ \neg b >) = sup(< a >) - sup(< a\ b >)$;

$sup(< \neg a\ b >) = sup(< b >) - sup(< a\ b >)$;

$sup(< b\ \neg a >) = sup(< b >) - sup(< b\ a >)$; and

$sup(< \neg b\ a >) = sup(< a >) - sup(< b\ a >)$.

Figure 6.2 is another example to show the meaning of $sup(ns)$ from the aspect of set theory. Given $ns = < a\ \neg b\ c\ \neg d\ e\ f >$, 1-$negMSS_{ns} = \{< a\ \neg b\ c\ e >, < a\ c\ \neg d\ e >, < a\ c\ e\ \neg f >\}$, $sup(ns) =| \{< a\ c\ e >\} | - |< a\ b\ c\ e > \cup < a\ c\ d\ e > \cup < a\ c\ e\ f >|$.

The above examples show that the negative containment definition is consistent with set theory.

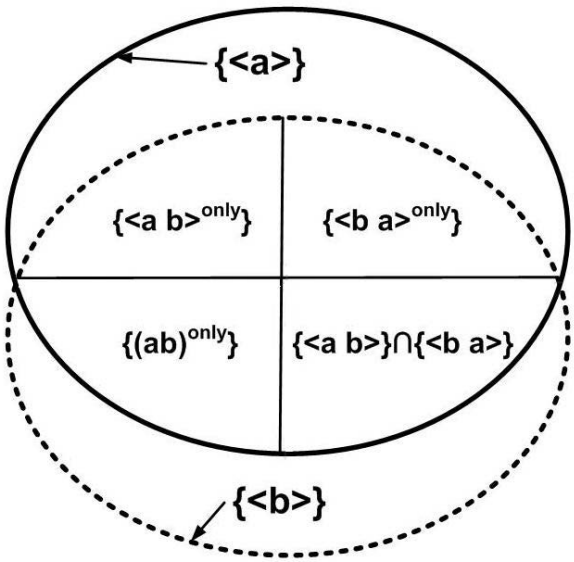


Figure 6.1: e-NSP Algorithm: the Intersection of $\{< a >\}$ and $\{< b >\}$

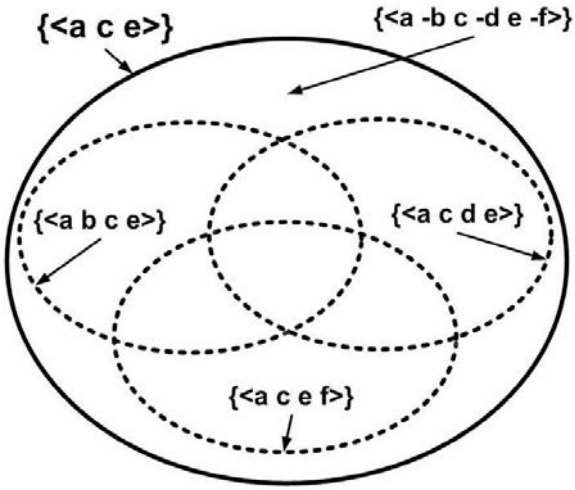


Figure 6.2: e-NSP Algorithm: the Meaning of $sup(< a\ \neg b\ c\ \neg d\ e\ \neg f >)$

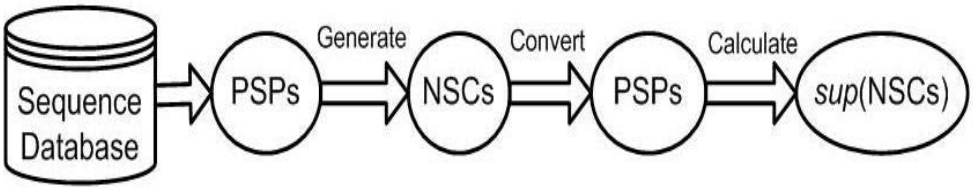


Figure 6.3: e-NSP Algorithm: Framework

6.2 e-NSP Algorithm

Figure 6.3 shows the framework and working mechanism of e-NSP, which consists of several steps as follows.

Step 1: Find all PSP from sequence database using any PSP Mining algorithm, such as GSP, PrefixSpan, SPADE, etc.;

Step 2: Generate NSC based on the PSP we obtained from Step 1, see Section 6.2.1;

Step 3: Use the information of corresponding PSP to calculate the support of NSC in Section 6.2.2;

We described a data structure and optimization for calculating the union set in Section 6.2.4. Section 6.2.5 shows e-NSP in detail, and its pseudocodes.

6.2.1 Negative Sequential Candidates Generation

The approaches of generating NSC in some other papers (Hsueh et al. 2008) (Zheng et al. 2009) are not compatible with the constraints of negative sequence in our approach, so we design an efficient approach to generate NSC. The basic idea of generating a NSC is to change any non-contiguous elements (not items) in a PSP to their negative ones. For a k-size PSP, its NSC are generated by changing any m non-contiguous element(s) to its (their) negative one(s).

Example 12. Given a PSP $< (ab)\ c\ d >$, it generates corresponding NSC including:

$m = 1, < \neg(ab)\ c\ d >, < (ab)\ \neg c\ d >, < (ab)\ c\ \neg d >$;

$m = 2, < \neg(ab)\ c\ \neg d >$.

Obviously, for all PSP in a sequence database, we can generate NSC that all satisfy the three constraints, which were described in Section 6.1.

6.2.2 Supports of Negative Sequences / Candidates

After obtaining NSC, the next step is to calculate supports of all NSC. Scanning the whole database is the only method to get the supports directly, as far as we know, but it is time consuming. Therefore, if we can verify whether a data sequence contains a NSC based only on the information of the PSP, without scanning the whole database, it will be more effective since we have already obtained all PSP in a previous step. In this section, we will discuss the method for calculating the supports of negative sequences / candidates using only identified positive patterns.

Given a *m-size* and *n-neg-size* negative sequence ns, for $\forall$1-$negMS_i \in$ 1-$negMSS_{ns}(1 \leqslant i \leqslant n)$, the support of ns in sequence database D is:

$$sup(ns) =\mid \{MPS(ns)\} - \cup_{i=1}^{n}\{PP(\text{1-}negMS_i)\} \mid \tag{6.1}$$

Equation 6.1 can be rewritten as:

$$\begin{aligned} sup(ns) &=\mid \{MPS(ns)\} \mid - \mid \cup_{i=1}^{n}\{PP(\text{1-}negMS_i)\} \mid \\ &= sup(MPS(ns)) - \mid \cup_{i=1}^{n}\{PP(\text{1-}negMS_i)\} \mid \end{aligned} \tag{6.2}$$

Example 13. $sup(< \neg a\ (bc)\ d\ \neg(cde)\ >) = sup(<(bc)d\ >) - \mid \{< a\ (bc)\ d >\} \cup \{< (bc)\ d\ (cde) >\} \mid$;

$sup(< \neg(ab)\ c\ \neg d >) = sup(< c >) - \mid \{< (ab)\ c >\} \cup \{< c\ d >\} \mid$.

If ns only contains a negative element, the support of ns is calculated by following equation:

$$sup(ns) = sup(MPS(ns)) - sup(PP(ns)) \tag{6.3}$$

Example 14., $sup(< (ab)\ \neg c\ d >) = sup(< (ab)\ d >) - sup(< (ab)\ c\ d >$)

Specially, for negative sequence $< \neg e >$,

$$sup(< \neg e >) =\mid D \mid -sup(< e >) \tag{6.4}$$

From equation 6.2 we can see that $sup(ns)$ can be easily calculated if we know $sup(MPS(ns))$ and $\mid \cup_{i=1}^{n}\{PP(1\text{-}negMS_i)\} \mid$. According to the constraints of NSC and the NSC generation approach discussed in Section 6.2.1, $MPS(ns)$ and $PP(1\text{-}negMS_i)$ are frequent. So $sup(MPS(ns))$ can be easily obtained by traditional algorithms.

The problem now is changed to how to calculate $\mid \cup_{i=1}^{n}\{PP(1\text{-}negMS_i)\} \mid$. Our approach is as follows. The *sid* of the tuples containing *PP(1-negMS$_i$)* are stored into set $\{$*PP(1-negMS$_i$)*$\}$, and then we calculate the union set of $\{$*PP(1-negMS$_i$)*$\}$. Because *PP(1-negMS$_i$)* are frequent, the *sid* of the tuples containing *PP(1-negMS$_i$)* can be easily obtained by well-known algorithms with minor modifications. For instance, we store the *sid* of the tuples containing *PP(1-negMS$_i$)* to $\{$*PP(1-negMS$_i$)*$\}$, while traditionally only the number of the tuples containing them is stored in mining PSP.

6.2.3 Negative Conversion Strategy and Proof

Theorem 1. Negative Conversion Strategy

Given a data sequence $ds =< d_1\ d_2 \ldots d_t >$, and $ns =< s_1\ s_2 \ldots s_m >$, which is an *m-size* and *n-neg-size* negative sequence, the negative containment definition can be converted as follows: data sequence ds contains negative sequence ns if and only if the two conditions hold: (1) $MPS(ns) \subseteq ds$; and (2) $\forall 1\text{-}negMS \in 1\text{-}negMSS_{ns}, PP(1\text{-}negMS) \nsubseteq ds$.

Example 15. Given $ds =< a\ (bc)\ d\ (cde) >$, 1) if $ns =< a\ \neg d\ d\ \neg d >$, $1\text{-}negMSS_{ns} = \{< a\ \neg d\ d >, < a\ d\ \neg d >\}$, then ds does not contain ns because $PP(< a\ \neg d\ d >) =< a\ d\ d > \subseteq ds$; 2) if $ns' =< a\ \neg b\ b\ \neg a\ (cde) >$, $1\text{-}negMSS'_{ns} = \{< a\ \neg b\ b\ (cde) >, < a\ b\ \neg a\ (cde) >\}$, then ds contains ns because $MPS(ns) =< a\ b\ (cde) > \subseteq ds \wedge PP(< a\ \neg b\ b\ (cde) >) \nsubseteq ds \wedge PP(< a\ b\ \neg a\ (cde) >) \nsubseteq ds$.

Proof of Theorem 1

Here we only prove that case c) in the negative containment definition is equivalent to the negative converting strategy, because cases a) and b) can be proved in the same way. In case c), condition "$(fse > 0 \wedge lsb = fse+1)$" says that d_{fse} and d_{lsb-1} are contiguous elements, so $PP(s_i)$ can't be contained between them. It is a special case of another condition "$(fse > 0 \wedge lsb > fse + 1) \wedge PP(s_i) \not\subseteq < d_{fse+1} \dots d_{lsb-1} >$". So we only need to prove that "$(fse > 0 \wedge lsb > fse + 1) \wedge PP(s_i) \not\subseteq < d_{fse+1} \dots d_{lsb-1} >$" is equivalent to negative converting strategy.

For $(s_i, id(s_i)) \in EidS^-_{ns}(1 \leqslant i \leqslant m)$, "$0 < fse$" means $MPS(<s_1\ s_2\ \dots\ s_{i-1}>) \subseteq <d_1\ \dots\ d_{fse}>$, and "$0 < lsb$" means $MPS(< s_{i+1} \dots s_m >) \subseteq < d_{lsb} \dots d_t >$. Since $fse < lsb$, $MPS(< s_1\ s_2 \dots s_{i-1}\ s_{i+1} \dots s_m >) \subseteq < d_1 \dots d_{fse}\ d_{lsb} \dots d_t >$, i.e., $MPS(ns) \subseteq ds$. On the other hand, if $MPS(ns) \subseteq ds$, for $\forall (s_i, id(s_i)) \in EidS^-_{ns}$, there must exist $0 < fse < lsb$ s.t. $MPS(< s_1\ s_2\ \dots s_{i-1} >) \subseteq < d_1\ \dots d_{fse} >$ and $MPS(< s_{i+1} \dots s_m >) \subseteq < d_{lsb} \dots d_t >$.

In addition, according to the definition of 1-neg-size maximum subsequence, $MPS(< s_1\ s_2\ \dots s_{i-1} >)$, $MPS(< s_{i+1} \dots s_m >)$ and s_i just construct a 1-neg-size maximum subsequence 1-negMS of s_i, so "$(fse > 0 \wedge lsb > fse + 1) \wedge PP(s_i) \not\subseteq < d_{fse+1} \dots d_{lsb-1} >$" also means $PP(1\text{-}negMS) \not\subseteq ds$. For $\forall (s_i, id(s_i)) \in EidS^-_{ns}$, "$(fse > 0 \wedge lsb > fse + 1) \wedge PP(s_i) \not\subseteq < d_{fse+1} \dots d_{lsb-1} >$" can be converted to: $\forall 1\text{-}negMS \in 1\text{-}negMSS_{ns}, PP(1\text{-}negMS) \not\subseteq ds$, and vice versa.

Corollary 1 proves that the negative containment problem is equivalent to the negative conversion problem.

6.2.4 e-NSP Data Structure and Optimization

In order to efficiently calculate the union set, we design a data structure to store the e-NSP related data. The data structure is shown in Table 6.1. Column one stores positive sequential patterns, column two holds its sup-

port, column three encloses $\{sid\}$ which is the set of the tuples that contain corresponding PSP.

Table 6.1: e-NSP: Data Structure and An Example

PSP	Support	{sid}
$<(ab)>$	2	{20,30}
$<(ab)\ c>$	1	{30}
...	...	...

The e-NSP data is stored in a hash table to identify PSP efficiently, as shown in the following pseudocode.

Function: Create hash table for calculating support of NSC.
Input: All PSP and their related information;
Output: PSP' hash table;

CreatePSPHashTable(PSP){
01. HashTable $PSPHash$ = new HashTable();
02. **for** (each pattern p in PSP)
03. $PSPHash$.put(p.HashCode(),p.support,p.size,p.sidSet);
04. **return** $PSPHash$;
}

In order to calculate the union set efficiently, we propose two other optimization methods as follows.

(1) When we calculate support of a NSC, we also utilize a hash table to accelerate search speed. For example, given a NSC ns=$< a\ \neg b\ c\ \neg d\ e\ \neg f >$, 1-$negMSSns$ = $\{< a\ \neg b\ c\ e >, < a\ c \neg d\ e >, < a\ c\ e \neg f >\}$, for each sequence 1-$negMS_i$ in 1-$negMSSns$, it is easy to get its positive partner PP(1-$negMS_i$). Then we search all $\{sid\}$ of PP(1-$negMS_i$) and add each

sid to the hash table if the *sid* is not available in the hash table. Finally, ns's support can be easily calculated by $sup(MPS(ns))$ minus the size of the new sid set, according to Equation 6.2. Compared with the performance using common array, the search speed with hash table is far more effecient.

(2) We assume that all data in the e-NSP data structure are stored in main memory. This is feasible in practice since the mainstream memory can reach gigabytes and above. We do not record the $\{sid\}$ of 1-size PSP because the equations do not need to calculate the union set of those $\{sid\}$ of 1-size PSP. Even in the worst situation in which the data are too big to fit completely into the main memory, by using the e-NSP candidate generation method for a k-size positive sequential pattern, the size range of negative elements in the corresponding NSC is $1 \rightarrow \lceil k/2 \rceil$, the size range of PSP used to calculate the support of these candidates is $(\llcorner k/2 \lrcorner + 1) \rightarrow (k-1)$, where $\llcorner k/2 \lrcorner$ is a maximum integer not more than $k/2$. Therefore, only $\{sid\}$s of $(\llcorner k/2 \lrcorner + 1) \rightarrow (k-1)$ size PSP are necessary to be put into main memory.

Example 16. (Continuation of Example 12) Given a PSP s, $size(s) = 5$, the range of neg-size of NSC based on the PSP is $1 \rightarrow 3$, and the size range of PSP used is $3 \rightarrow 4$. Specifically, when $neg\text{-}size(NSC) = 1$, the support of NSC can be calculated by Equation 6.2. When $neg\text{-}size(NSC) = 2$, we need to calculate the union set of 4-size PSP because $size(1\text{-}negMS_{NSC}) = 4$. When $neg\text{-}size(NSC) = 3$, we need to calculate the union set of 3-size PSP since $size(1\text{-}negMS_{NSC}) = 3$.

6.2.5 Pseudocode of e-NSP Algorithm

The e-NSP algorithm, as shown following, is proposed to mine for NSP. It consists of three key steps.

(1) Find all PSP from sequence database using any PSP Mining algorithm, such as GSP, PrefixSpan, SPADE, etc. All PSP and their *sid* sets are saved in a hash table $PSPHash$, in which PSP's hash code acts as the key code.

(2) For each PSP, generate NSC by the approach stated in Section 6.2.1.

(3) For each NSC ns, its support is calculated by Equation (6.1)-(6.2).

---

e-NSP Algorithm

Function: Run e-NSP algorithm to mine NSP;
Input: Sequence Database D and min_sup;
Output: Negative Sequential Patterns set NSP;

RunENSP(D, min_sup){
01. PSP = minePSP();
02. HashTable $PSPHash$ = CreatePSPHashTable(PSP);
03. **for** (each psp in PSP){
04. NSC = e-NSP_Candidate_Generation(psp);
05. **for** (each nsc in NSC){
06. **if** (nsc.size==1 && nsc.neg_size==1) {
07. nsc.support = |D| - $PP(nsc)$.support;
08. } **else if** (nsc.size>1 && nsc.neg_size==1){
09. nsc.support = $MPS(nsc)$.support-$PP(nsc)$.support;
10. } **else** {
11. 1-$negMSS_{nsc}$ = {1-$negMS_i$ | 1<=i<=nsc.neg_size};
12. HashTable $cHash$ = new HashTable();
13. **for** (i=1; i<=nsc.neg_size; i++) {
14. **for** (each sid in PP(1-$negMS_i$).sidSet){
15. **if** (sid.hashcode NOT IN $cHash$)
16. $cHash$.put(sid.hashcode(),sid);
17. nsc.support=$MPS(nsc)$.support-$cHash$.size();
18. }
19. **if** (nsc.support > min_sup)
20. NSP.add(nsc);

```
21.          }
22.       }
23.    }
24. }
25. return NSP;
}
```

6.2.6 An Example

The above sections introduce key concepts and components as well as the e-NSP algorithm for NSP mining. This section illustrates how to mine for NSP by an example. The sequence database is shown in Table 6.2 (Hsueh et al. 2008). In the example, we set *min_sup*=2 (to make the example easy to understand, we use the records count as *min_sup* here).

Table 6.2: e-NSP: Data Set for Example

Sid	Data Sequence
10	$< a\ b\ c >$
20	$< a\ (ab) >$
30	$< (ae)\ (ab)\ c >$
40	$< a\ a >$
50	$< d >$

The process is as follows.

(1) Mine PSP using one of the well-known algorithms, such as GSP, and fill in e-NSP data structure, which are shown as Table 6.3.

(2) Use the e-NSP Candidate Generation method to generate all NSC.

(3) Use Equations 6.1-6.2 to calculate the support of these NSC. The results are shown in Table 6.4, and the NSP are marked in bold.

From this example, we can see that $< a\ c >$ and $< a\ \neg c >$, $< a\ (ab) >$ and $< a\ \neg(ab) >$ are frequent patterns, but clearly not all of them can be used to

Table 6.3: e-NSP: Example Results - Positive Patterns

PSP	Support	Size	$\{sid\}$
$< a >$	4	1	-
$< b >$	3	1	-
$< c >$	2	1	-
$< a\ a >$	3	2	{20,30,40}
$< a\ b >$	3	2	{10,20,30}
$< a\ c >$	2	2	{10,30}
$< b\ c >$	2	2	{10,30}
$< (ab) >$	2	1	-
$< a\ b\ c >$	2	3	{10,30}
$< a\ (ab) >$	2	2	{20,30}

make decisions because some of patterns are misleading, which means some patterns may conflict with each other. For example, $< a\ c >$ is frequent, and $< a\ \neg c >$ is frequent as well, which one is useful for decision making? Some of them might not be useful. How to select the right patterns is one of our ongoing tasks, see Section 8.2.4.

6.2.7 Computational Complexity Analysis

Section 4.3.5 has described the computational complexity analysis of Neg-GSP and PNSP. In this section, we will talk about that of e-NSP.

e-NSP calculates the support value of each candidate by set operations, such as union, interaction and set difference. Since it will be much quicker than scanning the whole database, as what Neg-GSP and PNSP did, and e-NSP focus on smaller search space than that of PNSP and Neg-GSP, e-NSP will have higher performance.

According to the problem statement of e-NSP, negative candidates come from positive patterns. That is to say, the number of negative candidates would affected by the number of PSP. For a k-item positive pattern, it will

Table 6.4: e-NSP: Example Results - NSC and Supports

PSP	NSC	Related PSP	sup
$< a >$	$< \neg a >$	$< a >$	1
$< b >$	$< \neg \mathbf{b} >$	$< b >$	**2**
$< c >$	$< \neg \mathbf{c} >$	$< c >$	**3**
$< a\ a >$	$< \neg a\ a >$	$< a >, < a\ a >$	1
	$< a\ \neg a >$	$< a >, < a\ a >$	1
$< a\ b >$	$< \neg a\ b >$	$< b >, < a\ b >$	0
	$< a\ \neg b >$	$< a >, < a\ b >$	1
$< a\ c >$	$< \neg a\ c >$	$< c >, < a\ c >$	0
	$< \mathbf{a}\ \neg \mathbf{c} >$	$< a >, < a\ c >$	**2**
$< b\ c >$	$< \neg b\ c >$	$< c >, < b\ c >$	0
	$< b\ \neg c >$	$< b >, < b\ c >$	1
$< (ab) >$	$< \neg \mathbf{(ab)} >$	$< (ab) >$	**3**
$< a\ (ab) >$	$< \neg a\ (ab) >$	$< (ab) >, < a\ (ab) >$	0
	$< \mathbf{a}\ \neg \mathbf{(ab)} >$	$< a >, < a\ (ab) >$	**2**
$< a\ b\ c >$	$< \neg a\ b\ c >$	$< b\ c >, < a\ b\ c >$	0
	$< a\ \neg b\ c >$	$< a\ c >, < a\ b\ c >$	0
	$< a\ b\ \neg c >$	$< a\ b >, < a\ b\ c >$	1
	$< \neg a\ b\ \neg c >$	$< b >, < a\ b >, < b\ c >$	0

generate many 1-item to k-item candidates, as following:

$$\begin{aligned} N'_k &= n'_1 + n'_2 + n'_3 + \ldots + n'_k \\ &= k + \tfrac{k\times(k-1)}{2\times1} + \tfrac{k\times(k-1)\times(k-2)}{3\times2\times1} + \ldots + \tfrac{k\times(k-1)\times(k-2)\times\cdots\times1}{k\times(k-1)\times(k-2)\times\cdots\times1} \\ &= 2^k - 1 \end{aligned}$$

If there are m_1 1-item positive patterns, m_2 2-item positive patterns, ..., and m_k k-item positive patterns, the number of all candidates will be:

$N = m_1 \times (2^1 - 1) + m_2 \times (2^2 - 1) + \ldots + m_k \times (2^k - 1)$

Therefore, the computational complexity of e-NSP is $O(M \times 2^k)$, where M is the number of the positive patterns, and k is the length of the longest positive patterns.

6.3 Experiments and Evaluation

We conduct experiments on 14 synthetic and real datasets to compare the efficiency and scalability of e-NSP with two baseline approaches PNSP (Hsueh et al. 2008) and Neg-GSP (Zheng et al. 2009). We select PNSP and Neg-GSP because they are the only available algorithms that are comparable to our algorithm. To compare their performance, we adapt PSNP and Neg-GSP to follow the same definitions and constraints as stated in Section 6.1. In the comparison, all positive patterns are identified by GSP. NSP are further mined by e-NSP, PNSP and Neg-GSP. We conduct intensive experiments to compare the difference between three algorithms in terms of computational costs on different data sizes and data characteristics, and scalability. We also apply NSP to the detection of fraudulent claims in health insurance.

All algorithms are implemented in Java in a PC with Intel Core 2 CPU of 2.9GHz, 2GB memory and Windows XP Professional SP2. All algorithms are evaluated in terms of the execution time and scalability.

6.3.1 Data Sets

To describe and observe the impact of data characteristics on algorithm performance, we define the concept of data factors.

Data Factor

A data factor describes the characteristic of underlying data from a particular perspective. It was introduced in Section 1.3. The data factors include: *C, T, S, I, DB* and *N*

C: Average number of elements per sequence;

T: Average number of items per element;

S: Average length of maximal potentially large sequences;

I: Average size of items per element in maximal potentially large sequences;

DB: Number of sequences (= size of Database); and

N: Number of items.

Four source datasets are used for the experiments. They include both real data and synthetic datasets generated by IBM data generator (Agrawal & Srikant 1995). By partitioning the data, we obtain 14 datasets in total.

Dataset 1 (DS1), C8_T4_S6_I6_DB100k_N100, which means the average number of elements in a sequence is 8, the average number of items in an element is 4, the average length of a maximal pattern consists of 6 elements and each element is composed of 6 items average. The data set contains 100k sequences, the number of items is 100. The minimum number of elements in a sequence is 1, and the maximum number is 182.

We further adjust $DS1$ to generate 10 additional datasets, labelled as $DS1.x$ $(x = 1, \ldots, 10)$.

Dataset 2 (DS2), C10_T8_S20_I10_DB10k_N200, which means the average number of elements in a sequence is 10, the average number of items in an element is 8, the average length of a maximal pattern consists of 20 elements and each element is composed of 10 items average. The data set contains 10k sequences, the number of items is 200. The minimum number of elements in a sequence is 1, and the maximum number is 256.

Dataset 3 (DS3) is from UCI and consists of MSNBC.com anonymous web data about web page visits. Visits are recorded at the page category and are recorded in a temporal order. There are 989,818 records in the dataset. The average number of elements in a sequence is 4, and each element only has

one item. Its file size is 12M, which is relatively small since the dataset is shown in a special format without sequence id and element id. The minimum number of elements in a sequence is 1, and the maximum number is 323.

Dataset 4 (DS4) is real-application data about health insurance claim sequences. The data set contains 5,269 customers/sequences. The average number of elements in a sequence is 21. The minimum number of elements in a sequence is 1, and the maximum number is 144. The file size is around 5M.

6.3.2 Computational Cost

The execution time of mining NSP by the three approaches is shown in Figure 6.4. e-NSP takes less than 3% of the execution time of PNSP and Neg-GSP on all datasets. For example, e-NSP spends 2.7% to 1.6% execution time of PNSP on $DS3$ when min_sup decreased from 0.025 to 0.01. When min_sup is reduced to 0.01, PNSP and Neg-GSP take around one hour, but e-NSP takes less than one minute, because e-NSP only needs to "calculate" NSP support based on the *sid* sets of corresponding positive patterns, while PNSP and Neg-GSP have to re-scan the whole dataset.

The results about the maximum length and number of negative patterns are shown in Figure 6.5. It is difficult to get a reliable conclusion from them, since the datasets characteristics are not comparable. Therefore, we conduct dataset characteristics analysis in following section.

6.3.3 Dataset Characteristics Analysis

We analyze the dataset characteristics in terms of the above defined data factors to see the impact of the data factors (see Section 6.3.1) on the performance of e-NSP, compared to PNSP and Neg-GSP. We generate various types of synthetic datasets with different distributions.

Dataset *DS1* is extended to ten different datasets by tuning each factor, as shown in Table 6.5. For example, dataset *DS1.1* (C4T4S6I6. DB10k.N100)

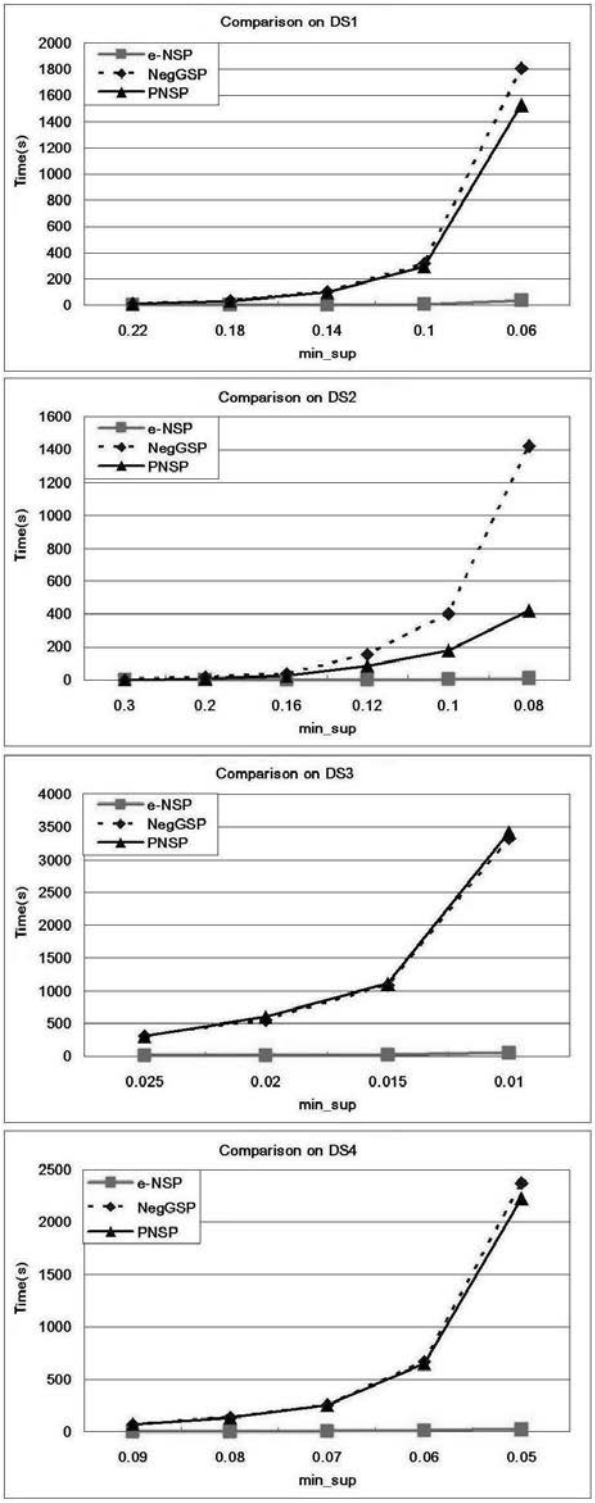


Figure 6.4: e-NSP Algorithm: Execution Time Comparison

is different to *DS1*(C8T4S6I6.DB10k.N100) on C factor, which means they have different average numbers of elements in a sequence. We mark the difference by underlining the distinct factor for each dataset in Table 6.5.

In Table 6.5, *t1*, *t2* and *t3* represent the execution time of Neg-GSP, PNSP and e-NSP correspondingly. We use *t3/t2* to show e-NSP's performance compared with PNSP. From the results (see Table 6.5), we can say that factors C, T and N seriously affect the performance of e-NSP, and factors S and I do not greatly affect it. When factor C is low, such as *DS1.1*, e-NSP works better than on datasets with big C, such as *DS1* and *DS1.2*. Similar results hold for T, such as *DS1* with small T, compared with *DS1.3* and *DS1.4* with big T. When N is high, such as in *DS1.9* and *DS1.10*, e-NSP works better than that with small N, such as in *DS1*.

Figure 6.6 shows a big-picture of the results in Table 6.5 in terms of the ratio of *t3/t2*. In the figure, $DS1.1$, $DS1.9$ and $DS1.10$ have very low ratios, which means e-NSP outperforms PNSP much. Generally speaking, on most of the 11 datasets, when *min_sup* is set at higher value, the ratio would be less.

6.3.4 Scalability Test

e-NSP calculates support based on the *sid* sets of corresponding positive patterns; its performance is affected by the size of *sid* sets. If the dataset is huge, it produces big *sid* sets. A scalability test is conducted to test e-NSP's performance on large datasets. Figure 6.7 shows the results of the three approaches on datasets $DS1$ and $DS3$ in terms of different data sizes: 10K to 100K data sequences are extracted from $DS1$, and 100k to 1,000k from $DS3$, by setting *min_sup* 0.12 on $DS1$, and 0.015 on $DS3$.

On $DS3$, when the sampled data size is increased to 1,000k, e-NSP takes 24 seconds to get the results. This is around 12 times the execution time of a sampled data size of 100k. This indicates an increase of 10 times the data size, corresponding to a 12 times increase in execution time. Moreover, compared to PNSP and Neg-GSP, e-NSP for dataset $DS1$ takes only around

Table 6.5: e-NSP: Experiments of Dataset Factors Analysis

ID	Dataset	min sup	NegGSP *t1*	PNSP *t2*	e-NSP *t3*	*t3/t2*
DS1	C8T4S6I6.DB10k.N100	0.04	1451.7	638.2	14.94	2.3%
		0.06	241.4	163.1	4.16	2.5%
		0.08	78.9	61.9	1.53	2.5%
DS1.1	**C4**T4S6I6.DB10k.N100	0.01	517.5	208.4	1.08	0.5%
		0.015	130.4	64.5	0.33	0.5%
		0.02	48.0	28.4	0.16	0.5%
DS1.2	**C12**T4S6I6.DB10k.N100	0.14	229.0	191.9	7.99	4.2%
		0.16	127.6	109.5	4.49	4.1%
		0.18	73.8	66.9	2.53	3.8%
DS1.3	C8**T8**S6I6.DB10k.N100	0.22	130.8	118.5	5.22	4.4%
		0.24	83.7	76.5	3.19	4.2%
		0.26	55.9	52.8	2.14	4.1%
DS1.4	C8**T12**S6I6.DB10k.N100	0.3	1205.2	969.3	57.55	5.9%
		0.4	133.2	123.5	6.75	5.5%
		0.5	23.6	23.0	1.06	4.6%
DS1.5	C8T4**S12**I6.DB10k.N100	0.04	1130.0	478.6	12.22	2.6%
		0.06	187.0	124.7	3.39	2.7%
		0.08	61.2	47.5	1.23	2.6%
DS1.6	C8T4**S18**I6.DB10k.N100	0.04	297.1	157.4	3.47	2.2%
		0.06	64.2	45.5	0.97	2.1%
		0.08	23.5	19.0	0.36	1.9%
DS1.7	C8T4S6**I10**.DB10k.N100	0.06	690.2	395.1	7.33	1.9%
		0.07	334.7	227.5	4.23	1.9%
		0.08	188.1	138.0	2.63	1.9%
DS1.8	C8T4S6**I14**.DB10k.N100	0.08	983.9	630.8	8.88	1.4%
		0.1	320.5	248.9	3.63	1.5%
		0.12	141.8	112.7	1.61	1.4%
DS1.9	C8T4S6I6.DB10k.**N200**	0.03	378.2	98.4	0.59	0.6%
		0.04	101.8	43.1	0.17	0.4%
		0.05	39.5	23.3	0.06	0.3%
DS1.10	C8T4S6I6.DB10k.**N400**	0.015	823.0	97.4	0.08	0.1%
		0.02	197.3	42.0	0.03	0.1%
		0.025	99.8	20.6	0.02	0.1%

2% of the time when the data size is 10k. It still takes around 2% when the dataset grows to 100k. A similar result holds for dataset $DS3$. The results show that the performance of e-NSP is robust on large datasets.

6.3.5 Experiments Summary

In summary, the experiments show that:

(1) e-NSP is very efficient, taking less than 3% of the execution time of the two baseline algorithms;

(2) e-NSP works very well for a small number of elements in a sequence, a small number of items in an element, and for a large number of itemsets. The length of patterns and the average number of items in an element of patterns are not sensitive to e-NSP; and

(3) The scalability test shows that the execution time of e-NSP has a linear relation with the number of data sequences. It proves that e-NSP can also perform well on large datasets.

6.4 Conclusions

In this chapter, we have proposed a simple but very efficient NSP mining algorithm: e-NSP. e-NSP is based on a formal and consistent concept, negative containment, which defines how a data sequence contains a negative sequence. e-NSP encloses a negative conversion strategy to convert the problem of whether a data sequence contains a negative sequence to the problem of whether a data sequence contains some of the corresponding positive sequences. Supports of NSC are then calculated based only on the corresponding PSP. Finally, a simple but efficient approach has been proposed to generate negative sequential candidates. e-NSP has been tested on both synthetic and real-world datasets and compared with NSP mining algorithms. The experimental results and comparisons on 14 datasets from data characteristics and scalability perspectives have clearly shown that e-NSP is much

more efficient than existing approaches. e-NSP offers a new strategy for efficiently mining large scale negative sequential patterns.

- Mining NSP is very challenging due to the large search space of negative candidates. Current NSP techniques rely on re-scanning of databases after identifying positive patterns. This has been shown to be very inefficient, and little progress has been made.
- In this chapter, stronger constraints of NSP makes the search space much smaller than what it is under Neg-GSP (Chapter 4) and GA-NSP (Chapter 5).
- Set theory can be used if there are clear boundaries of NSP constraints, just like what are defined in this chapter.

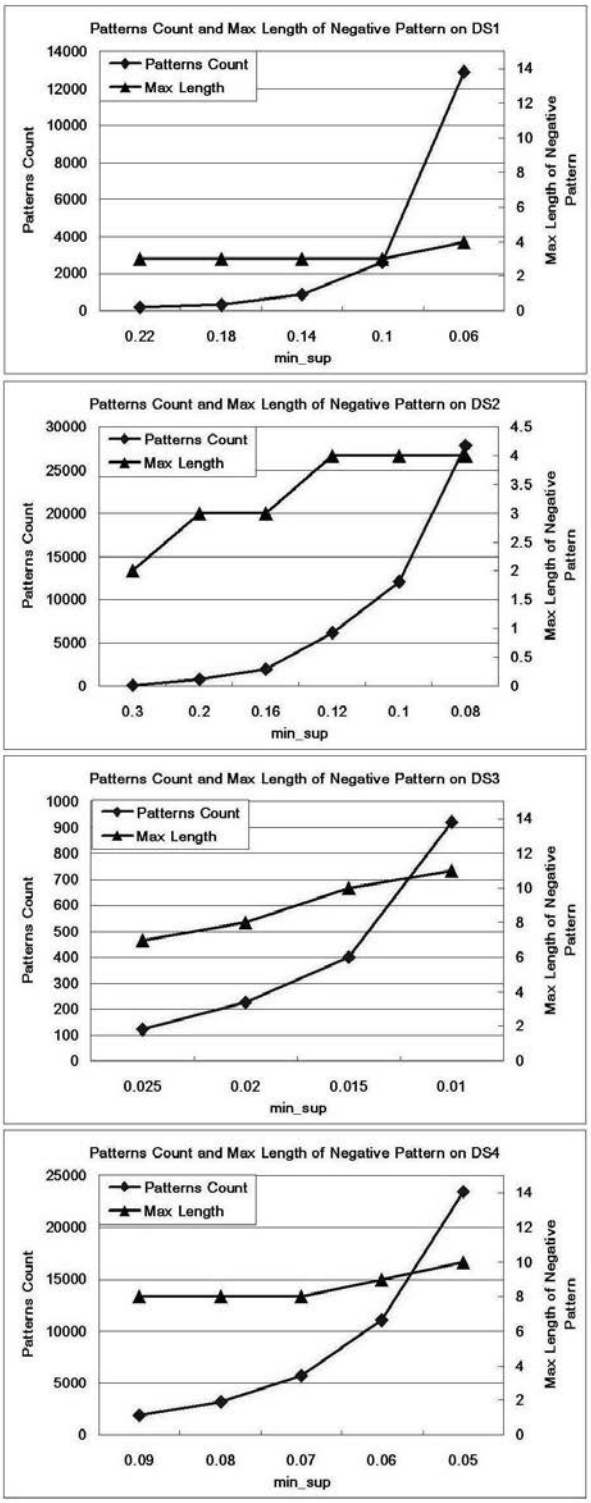


Figure 6.5: e-NSP Algorithm: Maximum Length and Patterns Counts

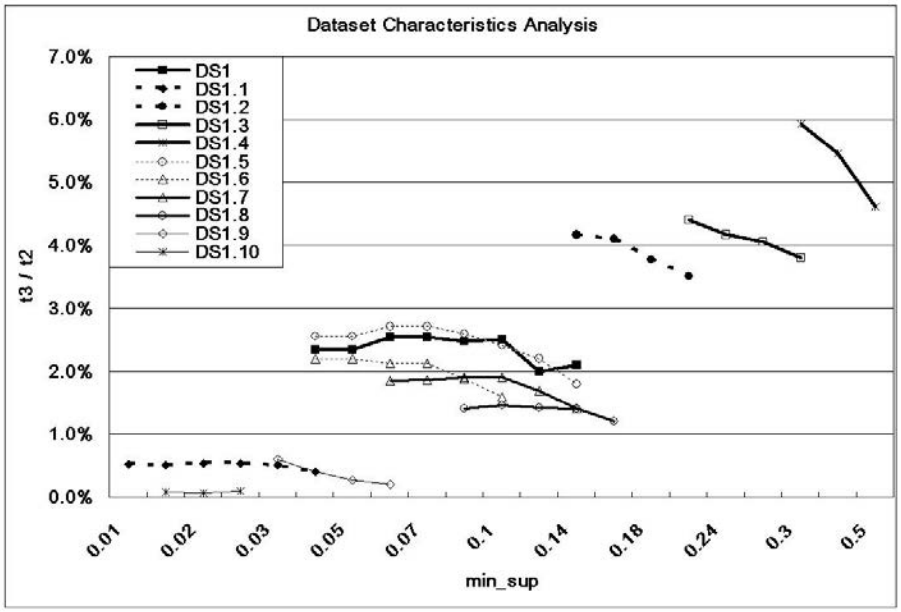


Figure 6.6: Dataset Characteristics Analysis

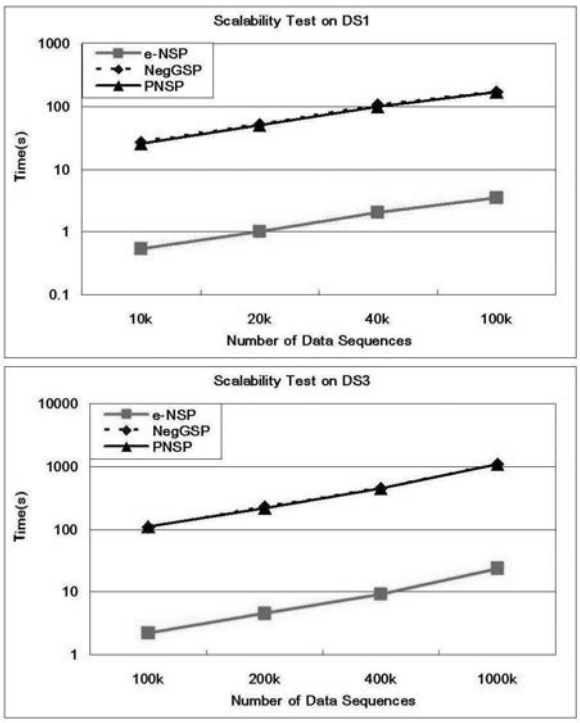


Figure 6.7: e-NSP Algorithm: Scalability Test

Chapter 7

Case Studies: NSP Applications

Two NSP mining applications in health insurance are introduced in the following. One tries to find over-service of ancillary services, and another is about fraud claim detection.

7.1 Case 1: Ancillary Services Over-service Analysis

This work is to explore relationships between chronic and acute conditions and over-service of ancillary services [1] [2]. In this work, chronic diseases are scoped as a kind of disease that is long-lasting, continuous or recurrent, and acute diseases are defined as diseases with a rapid onset or a short course. For different diseases, their common sequential patterns of ancillary services

[1] Zhigang Zheng, Ziye Zuo, Longbing Cao. Analysis of Ancillary Services with Chronic and Acute Diseases - By Sequence Analysis Methods. Technical Report of Industry Project, HIBIS Project, CMCRC & HCF, Feb 2010

[2] Uma Srinivasan. Industry Research Proposal: A note on Chronic Conditions and Acute Conditions - And Their Possible Relationship to Over-servicing of Physiotherapy and Related Allied Health Service Claims. HIBIS Project, CMCRC & HCF, 20 Oct 2009

usually demonstrate different trends.

For example, diabetes is a chronic condition, while hip replacement is considered an acute condition. Both people who got diabetes and those who had hip replacement usually undertake a long period of physiotherapy services [3], which belong to ancillary services. And they may have special trends or rules of physiotherapy services, i.e., most of the patients with hips replacement would go for physiotherapy services two weeks after their hospital treatments.

So it is supposed that we are able to explore the relationships between diseases and physiotherapy services and find out whether the physiotherapy services of a customer are over-serviced.

For each customer, we will track his physiotherapy activities for a period, i.e. 6 months, right after his hospital treatment and then try to find out trends or rules of physiotherapy services during the period.

First, we start with exploring all diseases which are most possibly related to physiotherapy. And then, we explore the distributions of physiotherapy services of the high-related diseases, and mine frequent patterns of physiotherapy services. After that, look for abnormal services, which don't fall into the normal patterns and are potential over-service. We also propose a scoring method to indicate customer's risk of being over-serviced.

7.1.1 Data Preparation

The first step is to find diseases which are highly related to physiotherapy services. We are interested in the diseases with which patients are most possible to go for physiotherapy service.

We deal with the high-related diseases one by one since different diseases have different trends of physiotherapy services.

We reconstruct each physiotherapy service sequence from day sequence

[3]Physiotherapy is a clinical health science and profession that aims to rehabilitate and improve people with movement disorders by using evidence-based, natural methods such as exercise, motivation, adapted equipment, education and advocacy.

to week sequence. For example, given a day sequence of customer's services $<$ 53 100 121 144 $>$, it means the customer got physiotherapy service 4 times after his hospital activity, with the first time of service being the 53rd days after the activity, second the 100th day, third and fourth time are the 121st and the 144th day correspondingly. It can be transformed to a week sequence $<$ *week*7 *week*14 *week*17 *week*20 $>$. The week sequence has higher granularity than day sequence.

7.1.2 PSP/NSP Mining by Neg-GSP Algorithm

In this step, we try to find frequent patterns on the week sequence dataset. Besides PSP, we are specially interested in NSP. For example, $<\neg$*week3* $\neg$*week4*$>$ means customers didn't go for service in the 3rd week or the 4th week. Since we focus on over-service analysis, getting NSP is more valuable for investigating the risk of over-service. After we get frequent PSP and NSP, we can utilize both of them to score risk levels of customers.

We adapt Neg-GSP to get NSP. As defined in Chapter 4, in Neg-GSP algorithm, we don't allow two or more than two adjacent negative elements, but in this case, we do need consider that because of nature of the data. When we consider two or more than two adjacent negative elements, it increases search space much more. Another aspect of the problem is that this case is much similar to association rule mining since the elements occurred by a special order, for example, *week*3 or $\neg$*week*3 is always after *week*1 and *week*2, and after $\neg$*week*1 and $\neg$*week*2 as well. And it will make the search space much smaller than it was expected. Therefore, on the combined impacts of the above two factors, the overall search space is not very huge, and we can still use Negtive-GSP to mine NSP efficiently in this application.

We modified the Neg-GSP algorithm from the following two aspects:

(1). Allow two or more than two adjacent negative elements while joining and generating candidates.

(2). Add another constraint in the joining step to make candidates follow the elements order. For example, it is not allowed to join and get candidate

like $< week3\ week2 >$ since $week3$ must occur after $week2$, and the same principle applies for negative items.

7.1.3 Risk Scoring

After getting all PSP and NSP in the above steps, we proposed a method to measure the distances between customer's service sequence and all patterns, including PSP and NSP. For each customer, we compared his/her service sequence with all PSN and NSP, and gave him/her a risk score to verify the possibility of over-service. If the sequence can match a pattern, the support count of the pattern is added to the score by Equation 7.1.

$$score = \sum_{i=1}^{n} sup(Pattern_i), \quad if\ service\ sequence\ matches\ Pattern_i \quad (7.1)$$

If a customer's score is very high, it means that he/she have quite normal patterns like most of other customers because he can match many frequent patterns. A customer with very low score is allocated with high level risk of over-service since his service activities are quite unique. We list the top 20 risk level customers with disease I18Z on Figure 7.1.

7.2 Case 2: Fraud Claim Detection

Frequent pattern mining can be applied to customer claims analysis. which contain many types of medical codes, such as CMBS, ICD10 and DRG codes. We are able to find out the frequent patterns of certain diseases or certain group of diseases with PSP mining.

Instead of considering only the occurrences of itemsets in sequences, the absence of itemsets are taken into account by NSP mining. Given a more concrete business example, in health insurance industry, there is a business rule, which is used to find claim fraud, it says "if a defibrillator prosthesis (A) has been charged, there should be a charge for a corresponding procedure

Support Score	Customer_deid	Day Sequence
0	89479	<18 20 22 24 27 29 31 36 39 41 43 50 52 55 57 59 62 67 71 73 76
0	1457235	<6 13 20 27 36 39 45 46 51 53 58 60 65 67 74 76 81 88 95 102 109
2844	391548	<69 73 76 78 80 83 85 94 97 99 101 104 106 111 113 118 122 125 1
2844	274857	<5 7 14 21 31 38 40 46 47 68 70 73 75 76 80 82 87 89 94 96 101 1
3432	28030	<10 17 24 31 38 45 52 58 65 72 79 86 93 100 107 114 128 135 142
3485	206884	<2 7 10 14 21 24 28 30 35 49 51 56 58 65 70 72 77 79 86 93 100 1
3519	84670	<16 20 23 37 44 51 58 65 72 79 86 -2 100 107 114 121 128 135 149
3549	911119	<1 2 5 7 10 13 17 19 20 26 28 30 34 36 41 43 48 49 55 57 62 64 6
3549	51589	<3 8 10 17 21 31 45 52 59 63 66 72 73 80 87 94 101 108 115 122 1
3549	911234	<3 4 6 26 34 35 47 48 49 52 53 54 55 56 60 61 62 66 67 68 69 70
3712	415724	<1 5 7 10 12 15 20 23 26 33 43 46 51 52 61 68 70 75 77 82 84 89
3712	1283396	<17 18 19 21 25 26 27 31 42 46 53 55 60 62 67 74 75 77 80 82 87
6393	154866	<66 71 73 76 80 83 - 94 101 104 108 115 118 122 125 129 132 136
6556	999664	<9 10 11 13 16 18 19 23 26 27 35 37 40 41 44 45 52 66 68 72 79 8
8954	808935	<80 88 - 93 94 100 104 107 108 109 110 111 114 115 116 117 118 1
9753	1565646	<6 8 11 12 13 14 16 20 21 23 24 25 27 30 31 48 49 54 56 57 61 62
10043	607069	<3 6 10 13 18 21 24 27 31 35 38 42 45 49 54 59 63 66 70 73 77 80
10095	321254	<2 3 4 6 9 11 13 16 20 41 44 51 55 62 68 74 81 88 95 102 111 120
10107	1017440	<4 7 9 11 14 16 18 21 23 25 28 30 32 35 37 39 42 44 46 49 53 58
10128	1771703	<2 3 5 9 10 11 15 16 17 18 21 22 25 28 29 30 31 32 35 36 37 38 3

Figure 7.1: Case Study: Example of Customers Risk Scoring

(B)". If a customer claims A, but not claim B, which is represented as $< A \neg B >$, it is a potential fraud claim.

7.2.1 Data Preparation

The customer claims data is composed of many fields, including customer id, separated date, three types of illness codes and service date, as shown in Figure 7.2. For a customer, his/her transactions are ordered by service date, and then can be used to generate a sequence. In the sample data of Figure 7.2, customer 10006 has a claim sequence *<(C2702, E1139, P500) (C2702, E1139, P505) (C2702, E1139, P505)>*, which is composed of three transactions, and each transaction has three items.

	customer_d	dte_separa	illness_code	icd10_princ	item_code	dte_service
	text	text	text	text	text	text
1	10006	2008-04-03	C2702	E1139	P500	2008-09-05
2	10006	2008-04-03	C2702	E1139	P505	2008-09-12
3	10006	2008-04-03	C2702	E1139	P505	2008-09-26

Figure 7.2: Samples of Customer Claims Dataset

7.2.2 A Fraud Scenario

Some Medical Codes Should Occur Together

In the health insurance service, the non-occurrence of some medical service codes in claim transactions may indicate a problem, such as fraud, in a service procedure, prothesis or a specific disease. For example, for certain patients with a special disease, a medical service code a usually occurs after a code b in their claim histories. If a customer claims a, but does not make a claim b prior to a, which is represented as $<\neg b\ a>$, the claim can be treated as a potential fraudulent claim. We therefore apply NSP mining to identify fraud in health insurance claims. The goal is to develop NSP that are associated with more than one non-occurring medical service code (that is, more than one negative item) in claims to detect claim fraud. As a result, we identify NSP, as the following, to detect claim fraud.

Fraud Claim: If a customer's medical claim sequence s satisfies the following conditions:

- $s=<i_1\ i_2\ \ldots\ i_n>$, where i_x $(1\leqslant x\leqslant n)$ is a positive item and represents one medical service code;
- s' is m-neg-size negative sequence, and the positive parter of s' P-$P(s')=s$;
  For example, when $m=1$, $s'=<i_1\ldots\neg i_x\ldots i_n>(1\leqslant x\leqslant n)$; when $m=2$, $s'=<i_1\ldots\neg i_x\ i_{x+1}\ldots i_{y-1}\ \neg i_y\ldots i_n>$ $(1\leqslant x<y\leqslant n)$;
- $0<sup(s')/sup(s)<min\_ratio$ (i.e. $min\_ratio = 0.02$)

the customer's claims are likely fraudulent, since those medical codes in s should be claimed together but they did not occur in the customer's claims.

The above negative claim patterns can be further converted into business rules, as in the following example.

If a patient claimed item 45530 for a rectus abdominis flap, item 45569 should also be claimed for the closure of the abdomen and reconstruction of

the umbilicus. [4] That is to say, 45569 should be claimed with 45530; if a patient's claim history shows <45530 ¬45569>, such a case is suspicious and should therefore be reviewed.

7.2.3 Fraud Claim Detection by e-NSP Algorithm

Using e-NSP algorithm as introduced in Chapter 6.2, we can find all PSP and its corresponding NSC. Based on them, it is not hard to find such fraudulent claims.

Step 1. We can find all PSP by PSP mining algorithms, such as GSP, PrefixSpan and so on. Their related information, such as support values, can be gotten as well.

Step 2. For all PSP, generate their corresponding NSC. For example, for PSP $< a\ b\ c >$, its corresponding NSC are $< \neg a\ b\ c >$, $< a\ \neg b\ c >$, $< a\ b\ \neg c >$ and $< \neg a\ b\ \neg c >$ according to the definitions of e-NSP algorithm, see Chapter 6.2 for details.

Step 3. Then, we modified e-NSP to find the fraud pairs of claims. In the process of original e-NSP, we find frequent patterns by calculating NSC's supports; in this application we need to compare all NSC's supports with their corresponding PSP's supports to find fraud pairs of claims, even if the NSC is not frequent.

For example, s=$< a\ b\ c >$ is a PSP, and based on s, we can get its corresponding NSC s', which could be $< \neg a\ b\ c >$, $< a\ \neg b\ c >$, $< a\ b\ \neg c >$ or $< \neg a\ b\ \neg c >$, and then all their supports are calculated correspondingly. If $0 < sup(s')/sup(s) < min\_ratio$, the claim sequences which support s' could be potential fraud.

In the original algorithm e-NSP, the target is to find frequent patterns from NSC. But hereby, we are only interested in the NSC with low ratio of

[4] (*http://www9.health.gov.au/mbs* 2011) CMBS code 45530 represents breast reconstruction using a latissimus dorsi or other large muscle or myocutaneous flap, including repair of secondary skin defects; 45569 represents closure of the abdomen with reconstruction of the umbilicus, with or without lipectomy.

$sup(s')/sup(s)$, and so s' is still get attention even if *sup(s')* is low.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

- Different from traditional PSP mining, NSP mining considers both positive and negative relationships between items. It doesn't necessarily follow the Apriori property, and the searching space of NSP mining is much larger than that of PSP mining.

- NSP mining can be categories into three types of problems, positive-first problem, negative-first problem and order-first problem. PSP mining problem is a special positive-first and order-first problem.

- Giving definitions and some constraints of NSP, this thesis proposes three innovative methods for mining NSP.

  The first algorithm is Neg-GSP, which is based on the PSP mining algorithm GSP, and we extend it to deal with negative cases. An effective pruning method to reduce the number of candidates is advised as well. Neg-GSP can find NSP effectively and efficiently by joining and pruning.

  The second method GA-NSP is based on Genetic Algorithm. The proposed method can find negative patterns efficiently since it gives more

priorities to the most frequent NSP. An effective dynamic fitness function and a pruning method are also provided to improve performance.

The third algorithm is e-NSP, which is based on the Set Theory. e-NSP mines for NSP by only involving the identified PSP, without re-scanning database. The basic working mechanism of e-NSP is as follows. First of all, negative containment is defined to determine whether a data sequence contains a negative sequence. It clearly defines the boundary of a data sequence set containing a NSC, and supports the application of the Set Theory in NSP mining. Secondly, an efficient approach is proposed to convert the negative containment problem to a positive containment problem. The supports of NSC are then calculated based only on the corresponding PSP. In this way, mining NSP does not need additional database scans, and the existing PSP mining algorithms can be used to mine for NSP. Finally, a simple but efficient approach is proposed to generate NSC.

- Comparing the NSP definitions of the above three methods, *Neg-GSP* and *GA-NSP* share the same definitions, *e-NSP* uses stronger constraints since it requires clear boundary to follow the Set Theory. When we compare their efficiency, *GA-NSP* algorithm slightly outperforms *Neg-GSP* in terms of the execution time, but it misses some of patterns in the complete result sets due to limitations of GA. Apparently, *e-NSP* is the most efficient and effective one since it doesn't need to scan datasets to calculate the support of NSP. Although adding stronger constraints makes the NSP set much smaller than what it is under the normal definitions, it is still very practicable while being adopted in some real-life applications.

- Neg-GSP has the same computational complexity as PNSP, but in practise, Neg-GSP can generate candidates more efficiently than PNSP.

- e-NSP is based on the Set Theory. It is quite effective and offers a new strategy for efficient mining of NSP in large datasets.

8.2 Future Work

8.2.1 Future Work of Current Topics

In our future research, we will explore post mining methods to find interesting patterns from discovered NSP. Since we often obtain a large amount of NSP, finding interesting and interpretable patterns from them is crucial in real-world applications.

Another potential research direction is to find more effective pruning methods that can reduce candidates and avoid generating unnecessary candidates.

To further improve the GA-NSP algorithm, we will focus on studying new measures including fitness functions, selection and crossover methods to make the algorithm more efficient and find more effective pruning strategies.

Using low bound and upper bound to calculate the support of NSP instead of precise support in e-NSP algorithm could possibly reduce the cost of computation. It could be used to prune invalid candidates as much as possible.

8.2.2 Order-First and Negative-First Problems

Since most of existing research work focuses on the positive-first problem, and seldom touches the order-first or negative-first problem, it will be an interesting topic for further research. The order-first problem is especially suitable for protein and gene sequence analysis since the protein and gene sequence data are always ordered by positions. If we take non-occurring items information into account, it will be able to find more interesting patterns and to improve the accuracy of classification and prediction.

8.2.3 Negative Sequential Pattern Classification

Sequence classification has a broad range of applications such as genomic analysis, information retrieval, health informatics, finance, and anomaly de-

tection (Xing, Pei & Keogh 2010). Because sequences do not have explicit features even with advanced feature selection techniques, and the number of potential features are still very high, sequence classification itself is a challenging task. If we consider negative itemsets in sequence classification, it will make the problem even more complicated. But since the topic will still take advantage of negative patterns, it is expected to get much higher accuracy by negative or combined classifiers.

8.2.4 Post Mining of Negative Sequential Pattern

Identifying interesting patterns or rules from the discovered NSP will be necessary when we got a huge amount of uninterpretable negative patterns. How to find interesting and interpretable rules from them is challenging and valuable in business applications. They may include:

(1) Single high-impact pattern. Find special patterns with high impact, high confidence or by some other measures.

(2) Comparable patterns or pattern groups. Find similarity or difference between two patterns or among pattern groups, and then get interpretable patterns by going through comparison. For example, given two patterns $<a\ \neg b\ c>$ and $<a\ b\ d>$, after comparing them, we may find out that the second element $<b>$, which follows $<a>$, actually has a high impact on the third element c or d.

(3) Eliminating unmeaning and unnecessary patterns. For example, if $<a\ \neg b>$ and $<a\ b>$ are both frequent, it will be much interesting to learn which one is actually more important or necessary.

Appendix A

Appendix: List of Publications

Papers Published

- Dong, X., Zheng, Z., Cao, L., Zhao, Y., Zhang, C., Li, J., Wei, W. & Ou, Y. (2011), e-nsp: efficient negative sequential pattern mining based on identified positive patterns without database rescanning, CIKM '11, pp. 825-830.

- Zheng, Z., Zhao, Y., Zuo, Z. & Cao, L. (2010), An efficient ga-based algorithm for mining negative sequential patterns, in 'Advances in Knowledge Discovery and Data Mining, PAKDD '10, Vol. 6118, p-p. 262-273.

- Zheng, Z., Zhao, Y., Zuo, Z. & Cao, L. (2009), Neg-GSP: An efficient method for mining negative sequential patterns, in 'Data Mining and Analytics', Vol. 101, pp. 63-67.

- Cao, L., Luo, D., Xiao, Y. & Zheng, Z. (2008), Agent collaboration for multiple trading strategy integration, 'Agent and Multi-Agent Systems: Technologies and Applications', Vol. 4953, Springer Berlin / Heidelberg, pp. 361-370.

Papers to be Submitted/Under Review

- Zheng, Z., Cao, L. Framework of Negative Sequential Pattern Mining and Genetic Algorithm Based Approach, to be submitted as a journal paper.
- Zuo, Z., Zheng, Z., Yin, J., Li, J. & Cao, L. (2011), Discriminative Customer Behavior Analysis by Efficient Emerging Sequential Pattern Mining, to be submitted as a conference paper.

Research Reports of Industry Projects

- Zhigang Zheng, Ziye Zuo and Longbing Cao. Analysis of Ancillary Services with Chronic and Acute Diseases - By Sequence Analysis Methods. HIBIS Project, CMCRC & HCF Australia, Feb 2010.
- Zhigang Zheng, Junfu Yin and Longbing Cao. Mining Discriminative Patterns Showing Significant Behavioural Difference between Lapse and Active Customers. AMP Pilot Data Mining Project, AMP Australia, Jan 2011.

Appendix B

Appendix: List of Symbols

The following list is neither exhaustive nor exclusive, but may be helpful.

$\sqsubset$	Contain
$\not\sqsubset$	Not contain
$\sqsubseteq$	Absolutely contain
$\not\sqsubseteq$	Not absolutely contain
1-negMS(s)	1-neg-size maximum subsequence of sequence s
$1\text{-}negMSS_s$	1-neg-size maximum subsequence set of sequence s
C_k	k-item candidates set
$C_{neg,k}$	k-item negative candidates set
$EE(e)$	Equivalent element of element e
$EidS_s$	Elements ID set of sequence s
$EidS_s^+$	Positive elements ID set of sequence s
$EidS_s^-$	Negative elements ID set of sequence s
$FSEorfse$	First subsequence ending position

I	Set of items
I^+	Set of positive items
I^-	Set of negative items
ind	Individual in population (Genetic Algorithm)
L_k	k-item sequential patterns set
$L_{neg,k}$	k-item negative sequential patterns set
$L_{pos,k}$	k-item positive sequential patterns set
$length(s)$	Length of sequence s
$LSBorlsb$	Last subsequence beginning position
$MEE(e)$	Maximum equivalent element of element e
$MES(s)$	Maximum equivalent sequence of sequence s
min_sup	Threshold of minimum support value
$MPS(s)$	Maximum positive subsequence of sequence s
pop	Population in Genetic Algorithm
$PP(s)$	Positive partner of sequence s
$RP(s)$	Reverse partner of sequence s
$S_{neg,k}$	k-item negative seed set
s_c	Candidate sequence
s_d	Data sequence
S_k	k-item seed set
$size(s)$	Size of sequence s

$Sub(s)$	Subsequence of sequence s
$sup(s_c)$	Support value of candidate sequence s_c

Bibliography

Agrawal, R., Imieliński, T. & Swami, A. (1993), Mining association rules between sets of items in large databases, SIGMOD '93, ACM, New York, NY, USA, pp. 207–216.

Agrawal, R. & Srikant, R. (1994), Fast algorithms for mining association rules, VLDB '94.

Agrawal, R. & Srikant, R. (1995), Mining sequential patterns, *in* 'ICDE '95.', pp. 3–14.

Alata, B. & Akin, E. (2006), 'An efficient genetic algorithm for automated mining of both positive and negative quantitative association rules', *Soft Computing - A Fusion of Foundations, Methodologies and Applications* **10**, 230–237.

Antonie, M.-L. & Zaïane, O. R. (2004), Mining positive and negative association rules: an approach for confined rules, PKDD '04, pp. 27–38.

Ayres, J., Flannick, J., Gehrke, J. & Yiu, T. (2002), Sequential pattern mining using a bitmap representation, KDD '02, ACM, New York, NY, USA, pp. 429–435.

Bäck, T. (1996), *Evolutionary Algorithms in Theory and Pratice*, Oxford University Press, New York, USA.

Baker, J. E. (1987), Reducing bias and inefficiency in the selection algorithm, *in* 'Proceedings of the Second International Conference on Genetic Al-

gorithms and their application', L. Erlbaum Associates Inc., Hillsdale, NJ, USA, pp. 14–21.

Brin, S., Motwani, R. & Silverstein, C. (1997), Beyond market baskets: generalizing association rules to correlations, SIGMOD '97, ACM, New York, NY, USA, pp. 265–276.

Chiu, D.-Y., Wu, Y.-H. & Chen, A. (2004), An efficient algorithm for mining frequent sequences by a new strategy without support counting, *in* 'ICDE 2004', pp. 375–386.

Dong, G. (2009), *Sequence Data Mining*, Springer-Verlag, Berlin, Heidelberg.

Dong, X., Zheng, Z., Cao, L., Zhao, Y., Zhang, C., Li, J., Wei, W. & Ou, Y. (2011), e-nsp: efficient negative sequential pattern mining based on identified positive patterns without database rescanning, CIKM '11, pp. 825–830.

El-Sayed, M., Ruiz, C. & Rundensteiner, E. A. (2004), Fs-miner: efficient and incremental mining of frequent sequence patterns in web logs, WIDM '04, ACM, pp. 128–135.

Han, J., Cheng, H., Xin, D. & Yan, X. (2007), 'Frequent pattern mining: current status and future directions', *Data Mining and Knowledge Discovery* **15**, 55–86.

Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U. & Hsu, M.-C. (2000), Freespan: frequent pattern-projected sequential pattern mining, KDD '00, ACM, New York, NY, USA, pp. 355–359.

Han, J., Pei, J. & Yin, Y. (2000), Mining frequent patterns without candidate generation, SIGMOD '00, ACM, New York, NY, USA, pp. 1–12.

Haupt, R. & Haupt, S. (1998), *Practical Genetic Algorithms*, Wiley, New York, NY, USA.

Hipp, J., Güntzer, U. & Nakhaeizadeh, G. (2000), 'Algorithms for association rule mining - a general survey and comparison', *SIGKDD Explor. Newsl.* **2**, 58–64.

Holland, J. H. (1975), *Adaptation in Natural and Artificial Systems*, The University of Michigan Press.

Hsueh, S.-C., Lin, M.-Y. & Chen, C.-L. (2008), Mining negative sequential patterns for e-commerce recommendations, *in* 'IEEE Asia-Pacific Services Computing Conference, 2008. APSCC'08', pp. 1213–1218.

http://en.wikipedia.org/wiki/Sharp-P-complete (2012).

http://www9.health.gov.au/mbs (2011).

Huan, J., Wang, W., Prins, J. & Yang, J. (2004), Spin: mining maximal frequent subgraphs from graph databases, KDD '04, ACM, pp. 581–586.

Hussain, F., Liu, H., Suzuki, E. & Lu, H. (2000), Exception rule mining with a relative interestingness measure, *in* 'Knowledge Discovery and Data Mining. Current Issues and New Applications', Vol. 1805, Springer Berlin / Heidelberg, pp. 86–97.

Inokuchi, A., Washio, T. & Motoda, H. (2000), An apriori-based algorithm for mining frequent substructures from graph data, *in* 'Principles of Data Mining and Knowledge Discovery', Vol. 1910 of *Lecture Notes in Computer Science*, pp. 13–23.

Jean-Francois Boulicaut, A. B. & Jeudy, B. (2000), Towards the tractable discovery of association rules with negations, *in* 'Proceedings FQAS00, Advances in Soft Computing series', pp. 425–434.

Kuramochi, M. & Karypis, G. (2001), Frequent subgraph discovery, *in* 'ICDM 2001', pp. 313–320.

Lin, N. P., Chen, H.-J. & Hao, W.-H. (2007), Mining negative sequential patterns, *in* 'Proceedings of the 6th Conference on WSEAS International Conference on Applied Computer Science', Stevens Point, Wisconsin, USA, pp. 654–658.

Mabroukeh, N. R. & Ezeife, C. I. (2010), 'A taxonomy of sequential pattern mining algorithms', *ACM Comput. Surv.* **43**, 3:1–3:41.

Masseglia, F., P., P. & Cicchetti, R. (2000), 'An efficient algorithm for web usage mining', *NETWORKING AND INFORMATION SYSTEMS JOURNAL* **2**, 571–604.

MathWorks (2011), 'User's guide: Using the genetic algorithm'.

Miller, B. L. & Goldberg, D. E. (n.d.), *Genetic Algorithms, Tournament Selection, and the Effects of Noise*.

Mitchell, M. (1996), *Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA.

Ouyang, W. (2009), Mining positive and negative weighted fuzzy association rules in large transaction databases, *in* 'Second International Symposium on Knowledge Acquisition and Modeling, 2009.', Vol. 2, pp. 269–272.

Ouyang, W., Huang, Q. & Luo, S. (2008), Mining positive and negative fuzzy sequential patterns in large transaction databases, *in* 'Fifth International Conference on Fuzzy Systems and Knowledge Discovery, 2008. FSKD '08.', Vol. 5, pp. 18–23.

Ouyang, W.-M. & Huang, Q.-H. (2007), Mining negative sequential patterns in transaction databases, *in* 'International Conference on Machine Learning and Cybernetics, 2007', Vol. 2, pp. 830–834.

Padmanabhan, B. & Tuzhilin, A. (1998), A belief-driven method for discovering unexpected patterns, *in* 'AAAI 98', AAAI Press, pp. 94–100.

Padmanabhan, B. & Tuzhilin, A. (2000), Small is beautiful: discovering the minimal set of unexpected patterns, KDD '00, ACM, New York, NY, USA, pp. 54–63.

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U. & Hsu, M.-C. (2004), 'Mining sequential patterns by pattern-growth: the prefixspan approach', *IEEE Transactions on Knowledge and Data Engineering* **16**(11), 1424–1440.

Pei, J., Han, J., Mortazavi-asl, B. & Zhu, H. (2000), Mining access patterns efficiently from web logs, *in* 'Knowledge Discovery and Data Mining. Current Issues and New Applications', Vol. 1805, pp. 396–407.

Sarracco, F. (2003), 'Hypergraphs and fast mining of association rules'.

Savasere, A., Omiecinski, E. & Navathe, S. (1995), An efficient algorithm for mining association rules in large databases, VLDB '95, pp. 432–443.

Savasere, A., Omiecinski, E. & Navathe, S. (1998), Mining for strong negative associations in a large database of customer transactions, *in* 'Proceedings of14th International Conference on Data Engineering, 1998.', pp. 494–502.

Scime, A. (2004), *Web Mining: Applications and Techniques*, IGI Publishing, Hershey, PA, USA.

Srikant, R. & Agrawal, R. (1996), Mining sequential patterns: Generalizations and performance improvements, *in* 'Advances in Database Technology EDBT '96', Vol. 1057, Springer Berlin / Heidelberg, pp. 1–17.

Suzuki, E. (1997), Autonomous discovery of reliable exception rules, *in* 'In Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)', AAAI Press, pp. 259–262.

Suzuki, E. & Shimura, M. (1996), Exceptional knowledge discovery in databases based on information theory, *in* 'In Proceedings of the Sec-

ond International Conference on Knowledge Discovery and Data Mining (KDD-96)', AAAI Press, pp. 275–278.

Teng, W.-G., Hsieh, M.-J. & Chen, M.-S. (2002), On the mining of substitution rules for statistically dependent items, ICDM '02, IEEE Computer Society, Washington, DC, USA.

Whitley, D. (1994), 'A genetic algorithm tutorial', *Statistics and Computing* **4**, 65–85.

Wu, X., Zhang, C. & Zhang, S. (2002), Mining both positive and negative association rules, ICML '02, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 658–665.

Wu, X., Zhang, C. & Zhang, S. (2004), 'Efficient mining of both positive and negative association rules', *ACM Trans. Inf. Syst.* **22**, 381–405.

Xing, Z., Pei, J. & Keogh, E. (2010), 'A brief survey on sequence classification', *SIGKDD Explor. Newsl.* **12**, 40–48.

Yan, X. & Han, J. (2002), gspan: graph-based substructure pattern mining, *in* 'ICDM 2002', pp. 721–724.

Yang, Z., Wang, Y. & Kitsuregawa, M. (2007), Lapin: Effective sequential pattern mining algorithms by last position induction for dense databases, *in* 'Advances in Databases: Concepts, Systems and Applications', Vol. 4443 of *Lecture Notes in Computer Science*, pp. 1020–1023.

Zaki, M. J. (2001), 'Spade: An efficient algorithm for mining frequent sequences', *Machine Learning* **42**, 31–60.

Zaki, M. J., Parthasarathy, S., Ogihara, M. & Li, W. (1997), 'New algorithms for fast discovery of association rules'.

Zhao, Y., Zhang, H., Cao, L., Zhang, C. & Bohlscheid, H. (2008), Efficient mining of event-oriented negative sequential rules, *in* 'IEEE/WIC/ACM

International Conference on Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT '08.', Vol. 1, pp. 336–342.

Zhao, Y., Zhang, H., Cao, L., Zhang, C. & Bohlscheid, H. (2009), Mining both positive and negative impact-oriented sequential rules from transactional data, *in* 'Advances in Knowledge Discovery and Data Mining', Vol. 5476, Springer Berlin / Heidelberg, pp. 656–663.

Zhao, Y., Zhang, H., Wu, S., Pei, J., Cao, L., Zhang, C. & Bohlscheid, H. (2009), Debt detection in social security by sequence classification using both positive and negative patterns, ECML PKDD '09, Springer-Verlag, Berlin, Heidelberg, pp. 648–663.

Zheng, Z., Zhao, Y., Zuo, Z. & Cao, L. (2009), Negative-gsp: An efficient method for mining negative sequential patterns, *in* 'Data Mining and Analytics', Vol. 101, pp. 63–67.

Zheng, Z., Zhao, Y., Zuo, Z. & Cao, L. (2010), An efficient ga-based algorithm for mining negative sequential patterns, *in* 'Advances in Knowledge Discovery and Data Mining, PAKDD' 2010', Vol. 6118, Springer Berlin / Heidelberg, pp. 262–273.