MINING FREQUENT MAX AND CLOSED SEQUENTIAL PATTERNS

by

Ramin Afshar
B.Sc., University of Alberta, Alberta, 2000

THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
May 2002

APPROVAL

Name: Ramin Afshar

Degree: Master of Science

Title of thesis: Mining Frequent Max and Closed Sequential Patterns

Examining Committee: Dr. Jiawei Han
Chair

Dr. Jiawei Han
School of Computing Science
Senior Supervisor

Dr. Ke Wang
School of Computing Science
Supervisor

Dr. Martin Ester
School of Computing Science
External Examiner

Date Approved: May 24, 2002

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project and extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

Mining Frequent Max, and Closed Sequential Patterns

Author:

(signature)

Ramin Afshar

(name)

July 23, 2002

(date)

:kj

Abstract

Although frequent sequential pattern mining has an important role in many data mining tasks, however, it often generates a large number of sequential patterns, which reduces its efficiency and effectiveness. For many applications mining all the frequent sequential patterns is not necessary, and mining frequent Max, or Closed sequential patterns will provide the same amount of information. Comparing to frequent sequential pattern mining, frequent Max, or Closed sequential pattern mining generates less number of patterns, and therefore improves the efficiency and effectiveness of these tasks.

This thesis first gives a formal definition for frequent Max, and Closed sequential pattern mining problem, and then proposes two efficient programs MaxSequence, and ClosedSequence to solve these problems. Finally it compares the results, and performance of these programs with two brute force programs designed to solve the same problems.

Dedication

To my parents

Acknowledgments

I would like to thank the following people:

- my senior supervisor Dr. Jiawei Han, whom without his ideas and comments this thesis would not be possible.
- my supervisor Dr. Ke Wang who introduced me to the field of data mining.
- the other members of my committee for their time.
- Homayoun Afshar for his help in MaxSequence.
- Jian Pei for his helps, and comments during my research.
- Behzad Mortazavi-Asl for his code, and his help.
- members of Database, and Data Mining lab in Simon Fraser University for their support.
- all the people in the School of Computing Science.
- I wish to thank Blue Martini Software for contributing the KDD Cup 2000 data.

Table of Contents

List of Algorithms

List of Tables

List of Figures

Chapter 1. Introduction

Association rule, and Frequent pattern, mining was proposed by Agrawal, Imielinski, and Swami in [7]. Different techniques, and algorithms have been proposed for solving this problem [12, 13, 14, 15]. Frequent pattern mining can produce a large number of frequent patterns. Different studies have been done to enforce constraints into mining process, to generate only the interesting patterns. Pei, and Han in [16] show which constraints can be pushed into the mining process to improve its efficiency. In [17] Wang, He, and Han propose a method to add support constraint for itemsets. Garofalakis in [18] shows how constraints can be incorporated into sequential pattern mining process using regular expressions.

An alternative to frequent pattern mining is frequent Closed pattern mining, that produces less number of patterns, and is proved to be as useful as frequent pattern mining for association rule mining [8]. Zaki, and Hsiao in [8] proposed an algorithm called CHARM for frequent Closed pattern mining. Pei, Han, and Mao later in [9] introduced CLOSET, an efficient algorithm for mining frequent closed patterns that outperformed CHARM.

Frequent sequence mining can also produce an undesirable large number of sequences. Two alternatives for frequent sequence mining are frequent Max, and Closed sequence mining. In this thesis first I give a formal definition for theses two problems, and then I introduce two efficient programs to solve them.

1.1. Motivation

Since its introduction, sequential pattern mining [1] has become an important data mining task, and it has been used in a broad range of applications, including the analyses of customer purchase behavior, disease treatments, Web access patterns, DNA sequences, and many more. The problem is to find all sequential patterns with higher, or equal

support to a predefined minimum support threshold in a data sequence database. The support of a sequential pattern is the number, or percentage, of data sequences in the database that contain that pattern. Different techniques, and algorithms have been proposed to improve the efficiency of this task [1, 2, 4, 5, 6, 11].

The sequential patterns generated by the sequential pattern mining program can be used as the input of another program to do a specific data mining task, for example frequent patterns can be used to generate association rules. It has been recognized that by decreasing the minimum support, the number of frequent sequential patterns can grow rapidly. This large number of frequent sequential patterns can reduce the efficiency, and effectiveness of the mining task. The efficiency is reduced, because of the large number of patterns generated in the first stage needed to be processed in the later stages of the mining task. The effectiveness is also reduced, because users have to go through a large number of elements in the result set to find useful information.

We define a sequence, $\mathcal{M}$, as a frequent Max sequence, if there does not exist any frequent sequence that is the proper superset of $\mathcal{M}$. Sequence C is a frequent Closed sequence, if there does not exist any frequent sequence that is the proper superset of C, and has the same support as C (more formal definitions are in the following chapter). For many mining tasks the set of Max, or Closed sequences (with less number of elements than the complete set of frequent sequences), can provide the same amount of information, as the complete set of frequent sequences. In these tasks using the Max, or Closed, sequences, can improve the efficiency by reducing the number of the sequences needed to be processed. This also can improve the effectiveness by reducing the redundancy in the final results. In this thesis I introduce two efficient programs to mine Max, and Closed frequent sequences in a sequence database. These programs are based on Prefix-Projected Pattern Growth, PrefixSpan [4]. They generate some candidate frequent sequences, sequences that have the potential to be Max, or Closed. These candidate frequent sequences can be seen as local Max, or Closed sequences. A set of Max, or Closed, frequent sequences found to this point, the result set, is kept, and every new candidate is compared to this set. If the new candidate is a Max, or Closed, subset of

a sequence in the result set, then we ignore the candidate, otherwise we delete all the sequences in the result set that are Max, or Closed, subset of the candidate, and add the candidate to the result set. This way the result set will hold the global Max, or Closed, frequent sequences after all the candidates are checked. As it was described the whole process can be broken down into two major processes. Generating candidate sequences, and checking the candidates with the current result set.

The process of generating the frequent sequences is done efficiently using the PrefixSpan. PrefixSpan is one of the recent algorithms for mining frequent sequential patterns, and it out performs previous sequential pattern mining algorithms, especially when mining for long sequential patterns. Mining, and using frequent Max, or Closed, sequential patterns become more important, and efficient when there are many long frequent sequences. Since checking the candidates with the result set is a time consuming process, generating as few candidates as possible, and also using techniques to speed up the checking process, can improve the performance of the programs drastically.

1.2. Thesis Organization

The rest of the thesis is organized as follows. In addition to problem definition chapter 2 also contains some information about different sequential pattern mining methods. Chapter 3 discusses the general problem of Max, and Closed sequence mining. In chapter 4 brute force algorithms for mining Max, and Closed sequential patterns are introduced. I introduce two efficient programs *MaxSequence*, and *ClosedSequence* for mining frequent Max, and Closed sequences in chapter 5. The experimental, and performance results are presented in chapter 6. Chapter 7 covers the conclusion, and some ideas for future study.

Chapter 2. Problem Definition and PrefixSpan [4]

In this chapter, I first define the problem of frequent Max, and Closed, sequential pattern mining, and then in section 2.2 I give some background information about sequential pattern mining, and some detailed explanation of PrefixSpan.

2.1. Problem Definition

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of literals, called *items*. Any non-empty subset of I, is called an *itemset*. A *sequence* is an ordered list of itemsets. A sequence s is denoted by $<s_1\ s_2 \ldots s_l>$, where s_i is an itemset, i.e., $s_i \subseteq I$ for $1 \leq i \leq l$. s_i is also called an *element* of the sequence, and denoted as $(x_1 x_2 \ldots x_m)$ where x_j is an item, i.e., $x_j \in I$ for $1 \leq j \leq m$. For simplicity, the brackets are omitted for elements with only one item. Since an element is a set, and the order of its items is not important, we assume the items in an element are ordered alphabetically. *Length* of a sequence is the number of instances of items in that sequence. A sequence with length l is called an *l-sequence*. A sequence $\alpha = <a_1\ a_2\ \ldots\ a_n>$ is called a *subsequence* of the sequence $\beta = <b_1\ b_2\ \ldots\ b_m>$, and denoted as $\alpha \sqsubseteq \beta$, if there exist integers $1 \leq j1 < j2 < \ldots < jn \leq m$ such that $a_1 \subseteq b_{j1}, a_2 \subseteq b_{j2}, \ldots, a_n \subseteq b_{jn}$. Sequence β is also called the *supersequence* of α.

A set of tuples $<sid, s>$, where sid is a *sequence identifier*, and s is a sequence, is called a *sequence database*. A tuple $<sid, s>$ is said to *contain* a sequence α, if α is a subsequence of s, i.e., $\alpha \sqsubseteq s$. The *support* of a sequence α in a sequence database S, is the number of tuples in S containing α, i.e., $support_S(\alpha) = |\{<sid, s> | (<sid, s> \in S) \wedge (\alpha \sqsubseteq s)\}|$. A sequence $\alpha = <a_1\ a_2\ \ldots\ a_n>$ with support of m, is also shown as $<a_1\ a_2\ \ldots\ a_n> : m$. A sequence α is called a *frequent sequential pattern* in sequence database S, if the number of tuples in S that contain α is greater than or equal to a given positive integer ξ, called *support threshold*, or *minimum*

support, i.e., $support_S(\alpha) \geq \xi$. A sequence α is called a *frequent Max sequential pattern* in sequence database S, if α is a frequent sequential pattern in S, and there exists no frequent sequential pattern β in S, such that β is a proper supersequence of α. A sequence α is called a *frequent Closed sequential pattern* in sequence database S, if α is a frequent sequential pattern in S, and there exists no frequent sequential pattern β in S, such that (1) β is a proper supersequence of α, and (2) every tuple containing α also contains β, i.e., $support_S(\alpha) = support_S(\beta)$. The problem of *sequential pattern mining* is to find the complete set of frequent sequential patterns in a sequence database, for a given minimum support threshold. Given a sequence database S, and a minimum support threshold ξ, the problem of *sequential Max, or Closed, pattern mining* is to find the complete set of frequent Max, or Closed, sequential patterns in S. It is obvious that for any given sequence database, and support threshold the complete set of frequent sequences $\mathcal{F}$, and the complete set of frequent Max, and Closed, sequences $\mathcal{M}$, and C, the following relation holds:

$$\mathcal{M} \subseteq C \subseteq \mathcal{F}.$$

Given two sequences $\alpha = < a_1\ a_2\ \ldots\ a_n >$, and $\beta = < b_1\ b_2\ \ldots\ b_m >$, where $m \leq n$. β is called a *prefix* of α if and only if:

1. $a_i = b_i$ for $i \leq m - 1$;
2. $a_m \subseteq b_m$;
3. All the items in $(a_m - b_m)$ are alphabetically after those in b_m.

Given sequence α, and a subsequence of it, β. Another subsequence of α, α' is a *projection* of α with respect to prefix β if and only if:

1. α' has prefix β
2. No supersequence of α', α'' exists such that α'' is a subsequence of α, and also has prefix β.

2.2. Sequential pattern mining

Sequential pattern mining is an important data mining task, and different algorithms have been proposed to perform this task efficiently. The problem is to find all sequential patterns with higher, or equal support to a predefined minimum support threshold in a data sequence database. In this section we are going to discuss some of the proposed algorithms for this task. Since prefix projection is the main idea behind the MaxSequence, and ClosedSequence programs, we are going to discuss PrefixSpan in more detail.

2.2.1. Apriori Based Algorithms

Agrawal, and Srikant in [1] introduced the sequential pattern mining problem, and three algorithms to solve it. Among these algorithms *AprioriAll* was the only one to mine the complete set of frequent sequential patterns. Later in [2] they proposed the *GSP* (Generalized Sequential Patterns) algorithm, for solving this problem. GSP outperforms AprioriAll by up to 20 times. Both of these algorithms are based on the priori heuristics. The priori heuristics was proposed in [7] for association rule mining, and states that the sub patterns of a frequent pattern should also be frequent. Using this heuristic AprioriAll, and GSP narrow down the search space for frequent sequential patterns drastically. The mining process begins by scanning the database, and finding all the frequent items (length 1 frequent sequences). The mining process continues, considering priori heuristics, like this. Having all the length ℓ frequent sequences, these algorithms, considering priori heuristic, generate all the length $\ell + 1$ possible frequent sequences, and by scanning the database count the actual support for these sequences. Having the length $\ell + 1$ frequent sequences, this process can be repeated to get the $\ell + 2$ length frequent sequences.

For mining length ℓ frequent sequences in these Apriori based algorithms, ℓ scans of database is necessary. This can be very expensive for mining long frequent sequential patterns.

2.2.2. SPADE

Zaki in [6] proposed another approach for mining frequent sequential patterns, called *SPADE* (Sequential Pattern Discovery using Equivalence classes). This approach uses vertical database format, and using lattice search techniques, and join operations, mines the frequent sequential patterns. In this approach for each item a vertical id-list is created. Each list contains the sequence identifiers of the sequences this item appeared in, and their corresponding time stamps. By performing temporal joins on these id-lists all of the frequent sequential patterns can be enumerated. Decomposing the original search space into smaller subspaces using lattice-theoretic approach reduces search space in this method.

Unlike Apriori based algorithms this method does not make multiple scans of the database, and it can mine all the frequent sequences in three database scans.

2.2.3. FreeSpan

Han, et al. introduced the *FreeSpan* (Frequent pattern-projected Sequential pattern mining) in [5]. FreeSpan using frequent items, projects the sequence database into projected databases. Each projected database is then recursively projected further. The sizes of the projected databases often decrease rapidly, and these smaller databases are easier to work with. This method is considerably faster than Apriori based methods. The problem of this method is that the same sequences can be duplicated in many projected databases. In a later work [2] Pei, et al. introduced *PrefixSpan*. PrefixSpan not only eliminates the duplicated sequence problem, but also out performs previous sequential pattern mining algorithms, especially when mining for long sequential patterns.

Since mining, and using frequent Max, or Closed, sequential patterns become more important, and efficient, when there are many long frequent sequences generated, the PrefixSpan is a good choice to be used for Max, and Closed sequence mining. In next section we will study this method in more detail.

2.2.4. PrefixSpan

PrefixSpan [4] is a recently proposed method for mining frequent sequential patterns. Unlike Apriori-based algorithms, such as GSP [2], that mine frequent sequential patterns by candidate generation, PrefixSpan uses prefix projection to mine the complete set of frequent sequential patterns. Here is a brief description on how does PrefixSpan work. For more detailed information, I refer the reader to [4].

Given sequence database S, and minimum support of ξ, PrefixSpan performs the following steps to mine the frequent sequential patterns.

Step 1: Scan S once, and find all the frequent items. These frequent items are frequent sequential patterns with length one.

Step 2: According to the length-1 frequent sequences found in the first step, the complete set of frequent sequential patterns can be divided into different subsets, one subset for each length-1 frequent sequence.

Step 3: Each subset, in second step, can be mined by constructing its corresponding postfix *projected database*, and mining it recursively (a frequent item's, e, projected database is the set of postfixes of e in the original database).

PrefixSpan uses recursion to mine the frequent sequences. First frequent items in the database are found, and then for each frequent item a projected database is created (the

prefix of the projected database is a frequent sequence). This process is repeated for each projected database, until the projected database contains no frequent items, which in that time the recursion for that branch ends, and the execution returns to the calling procedure. Lets call this end of the recursion branch as a *backtrack*. Algorithm 2.1 is from [4], and shows how does PrefixSpan work.

Input: A sequence database S, and minimum support threshold ξ.
Output: The complete set of sequential patterns.
Method: Call **PrefixSpan**($<>$, 0, S)

Subroutine PrefixSpan(α, ℓ, $S|_\alpha$)
Parameters: α: a sequence; ℓ: the length of α; $S|_\alpha$: the α-projected database, if $\alpha \neq <>$, otherwise the sequence database S.
Method:
Scan $S|_\alpha$ once, find the set of frequent items b such that:
b can be assembled to the last element of α to form a sequence;
or $<b>$ can be appended to α to form a sequence.

For each frequent item b, append it to α to form a sequence α', and output α';

For each α', construct α'-projected database $S|_{\alpha'}$, and call **PrefixSpan**(α', l + 1, $S|_{\alpha'}$).

Algorithm 2.1 PrefixSpan, from [4].

In the following chapters I describe two programs for mining frequent Max, and Closed sequential patterns. These programs use PrefixSpan as their basis to generate candidate frequent Max, and Closed sequences.

Chapter 3. Mining frequent Max, and Closed Sequences

As it was mentioned earlier one way of mining frequent Max, or Closed, sequential patterns is to break down the process into two following major tasks:

1. Generating candidate sequences. The set of candidate sequences should be a superset of the complete set of Max, or Closed, sequences. Obviously the closer the number of sequences in the candidate set to the number of sequences in the Max, or Closed, sequence set, the better the candidate set.

2. Checking the candidates, and keeping the Max, or Closed, sequences as the result set. As the candidates are generated, we need to check them with the previously generated candidates, and keep only the Max, or Closed, ones. For any given candidate sequence α, we need to check, if there exists a sequence β, in the result set, such that α is Max, or Closed, subsequence of β. If yes, we ignore α, otherwise we delete all the sequences, which are Max, or Closed, subsequences of α from the result set, and add the sequence α to the result set. This way we guarantee that after checking all of the candidates, the result set will contain the complete set of Max, or Closed, sequences.

The described Max, or Closed, sequence mining process is shown in algorithm 3.1.

In the following chapters, two approaches for mining Max, or Closed, sequences are introduced. These approaches are similar in the way, that they do the mining following the two mentioned steps, but they are different in how they perform these steps. The first method, called the *Naïve* approach, is introduced in chapter 4. In chapter 5 the more efficient methods *MaxSequence*, and *ClosedSequence* are introduced. In chapter 6 the results, and performance of these methods are discussed.

Input: A sequence database S, and minimum support threshold ξ.
Output: The complete set of Max, or Closed sequential patterns.
Method:

```
ResultSet = { }
aSequence = GetFirstCandidate( )
While aSeqence is not empty
    AddToResultset( Resultset, aSequence )
    aSequence = GetNextCandidate( )
Output Resultset.
```

Subroutine AddToResultset(aSet, aSequence)
Parameters: aSet: is a set of sequences; aSequence: is a sequence.
Method:

```
IsSuper = False
For each sequence aSeq in aSet
    If IsSuper == False
        If aSeq is Max, or Closed, supersequence of aSequence
            Exit
    If aSequence is Max, or Closed, supersequence of aSeq
        Remove aSeq from aSet
        IsSuper = True
Add aSequence to aSet.
```

Function GetFirstCandidate ( )
Returns the first candid sequence (will be discussed for different methods).

Function GetNextCandidate ( )
Returns the Next candid sequence (will be discussed for different methods).

Algorithm 3.1 Mining Max, or Closed sequential patterns.

Chapter 4. Naïve Approach

Lets first discuss a simple, brute force, approach for mining Max, or Closed, sequences. I call this approach the naïve approach. This approach is based on the two step process mentioned in previous chapter, and performs these tasks as follows:

1. Since the complete set of Max, and Closed, sequences is the subset of the complete set of frequent sequences, we can use the complete set of frequent sequences as our candidate set. I used PrefixSpan for generating the frequent sequences in this method.

2. In this method we use a list structure as our result set. For any given candidate sequence α, starting from the first sequence in the list, we check all the sequences in the list to see, if there exists a sequence β, in the list, such that α is Max, or Closed, subset of β. As soon as we find a Max, or Closed, superset of α we stop checking sequences in the list, ignore α, and move to next candidate. If there is no Max, or Closed, superset of α in the list, we scan the list for subsets of α, and delete them from the list, and finally we add α to the list. After repeating these steps for all the candidates, the list will contain the complete set of frequent Max, or Closed, sequences.

For example consider the sequence database S shown in table 4.1. For support threshold value of 0.5, this database contains thirteen frequent sequences.

Sequence_id	Sequence
10	$<a_1\ a_2\ a_3\ a_4\ a_5>$
20	$<a_2\ a_4\ a_5>$
30	$<a_1\ a_3\ a_5>$

Table 4.1 A sequence database.

These sequences are shown on the middle column of table 4.2 in the order they are generated using PrefixSpan. In this example we take the set of all frequent sequences as our candidate set for Max sequence mining. The right column on each row of table 4.2 shows the result set for this process before checking the generated sequence with the result set. At the beginning, row 1, the result set is empty. After generating $<a_1>$, it is checked with the result set, since it is not a subsequence of any sequence in the result set it is added to the result set. This is the case for next four generated sequences $<a_2>$ to $<a_5>$. After adding $<a_5>$ the result set will contain elements shown in row 6. In this stage sequence $<a_1\ a_3>$ is generated, and it is checked with the result set. A supersequence of $<a_1\ a_3>$ does not exist in the result set, so we check for its subsequences in the set, $<a_1>$, and $<a_3>$ are subsequences of $<a_1\ a_3>$, so they are deleted from the result set, and $<a_1\ a_3>$ is added to the set. This continues till the last frequent sequence, $<a_4\ a_5>$, is generated. This sequence is checked with the result set, and sine the sequence $<a_2\ a_4\ a_5>$ in the result set is a supersequence of $<a_4\ a_5>$, it is not added to the result set. After this since there are no more frequent sequences, the result set contains the complete set of Max sequences.

Row	Frequent Sequences	Max Sequence Result Set
1	$<a_1>$	$\{\}$
2	$<a_2>$	$\{<a_1>\}$
3	$<a_3>$	$\{<a_1><a_2>\}$
4	$<a_4>$	$\{<a_1><a_2><a_3>\}$
5	$<a_5>$	$\{<a_1><a_2><a_3><a_4>\}$
6	$<a_1\ a_3>$	$\{<a_1><a_2><a_3><a_4><a_5>\}$
7	$<a_1\ a_5>$	$\{<a_2><a_4><a_5><a_1\ a_3>\}$
8	$<a_1\ a_3\ a_5>$	$\{<a_2><a_4><a_1\ a_3><a_1\ a_5>\}$
9	$<a_2\ a_4>$	$\{<a_2><a_4><a_1\ a_3\ a_5>\}$
10	$<a_2\ a_5>$	$\{<a_1\ a_3\ a_5><a_2\ a_4>\}$
11	$<a_2\ a_4\ a_5>$	$\{<a_1\ a_3\ a_5><a_2\ a_4><a_2\ a_5>\}$
12	$<a_3\ a_5>$	$\{<a_1\ a_3\ a_5><a_2\ a_4\ a_5>\}$
13	$<a_4\ a_5>$	$\{<a_1\ a_3\ a_5><a_2\ a_4\ a_5>\}$
14		$\{<a_1\ a_3\ a_5><a_2\ a_4\ a_5>\}$

Table 4.2 An example of Max sequence generation.

The algorithm for Naïve approach is shown in algorithm 4.1.

The complete set of frequent sequences is the upper bound for the candidate set, and since the tests performed, for each candidate, in the second step are very time consuming, an efficient mining method should be able to generate less number of candidates than the complete set of frequent sequences. Also using a list structure in the second step is not efficient, because for each candidate, we have to consider all the sequences in the list, and as the mining process continues the number of sequences in the list can become very large.

In following chapter we will see how we can improve the performance of the Max, and Closed, sequence mining, by generating less number of candidates, and also by improving the performance of the second, checking, step of the process. Also later in the results chapter the results, and performance of the newly developed methods will be compared with the results, and performance of the naïve approach.

Input: A sequence database S, and minimum support threshold ξ.
Output: The complete set of Max, or Closed sequential patterns.
Method:

ResultSet = { }
Call **NaïveApproach**(<>, 0, S, Resultset)
Output Resultset.

Subroutine NaïveApproach(α, l, $S|_\alpha$, Resultset)
Parameters: α: a sequence; l: the length of α; $S|_\alpha$: the α-projected database, if $\alpha \neq <>$, otherwise the sequence database S; Resultset: a set of sequences.
Method:

Scan $S|_\alpha$ once, find the set of frequent items b such that:

b can be assembled to the last element of α to form a sequence;
or $<b>$ can be appended to α to form a sequence.

For each frequent item b, append it to α to form a sequence α', and call **AddToResultset**(Resultset, α');

For each α', construct α'-projected database $S|_{\alpha'}$, and call **NaïveApproach**(α', l + 1, $S|_{\alpha'}$, Resultset).

Subroutine AddToResultset(aSet, aSequence)
Parameters: aSet: is a set of sequences; aSequence: is a sequence.
Method:

```
IsSuper = False
For each sequence aSeq in aSet
    If IsSuper == False
        If aSeq is Max, or Closed, supersequence of aSequence
            Exit
    If aSequence is Max, or Closed, supersequence of aSeq
        Remove aSeq from aSet
        IsSuper = True
Add aSequence to aSet.
```

Algorithm 4.1 Naive Approach for Max, or Closed, sequence mining.

Chapter 5. MaxSequence, and ClosedSequence

In this Chapter I propose two programs *MaxSequence*, and *ClosedSequence* for efficient mining of Max, and Closed sequences. These programs follow the outline described at chapter 3 (outline used by the Naïve approach), but they do the steps involved in more efficient ways. In section 5.1 I describe how these programs manage to do the mining process with generating less number of candidate sequences. How these programs perform the candidate-checking step is discussed in section 5.2. Section 5.3 describes another optimization method for Max sequence mining called String Elimination.

5.1. Generating Candidates

One approach to improve the performance of Max, or Closed, sequence mining is to reduce the number of candidates generated during the mining process. In ideal case the cardinality of the candidate set should be equal to the cardinality of the result set. This means that any candidate generated, is a Max, or Closed, sequence. So the lower bound for the cardinality of the candidate set is the cardinality of the result set. On the other hand for any given sequence database, and support threshold the complete set of frequent sequences $\mathcal{F}$, and the complete set of frequent Max, and Closed, sequences $\mathcal{M}$, and $\mathcal{C}$, have the following relation $\mathcal{M} \subseteq \mathcal{C} \subseteq \mathcal{F}$. From this we can see that $\mathcal{F}$ can be used as a candidate set for mining $\mathcal{M}$, and $\mathcal{C}$. In fact the cardinality of $\mathcal{F}$ is the upper bound for the cardinality of candidate set. In many cases the cardinality of $\mathcal{F}$ can be much bigger than the cardinality of $\mathcal{M}$, or $\mathcal{C}$, and this suggests that choosing $\mathcal{F}$ as candidate set is not a very good choice. For example as an extreme case consider the sequence database $\mathcal{S}$ given in Table 5.1, and the minimum support of 1 (i.e., every occurrence is frequent).

Sequence_id	Sequence
10	$< a_1\ a_2 \ldots\ a_{100} >$
20	$< a_1\ a_2\ \ldots\ a_{50} >$

Table 5.1 An extreme sequence database.

For given minimum support this sequence dataset has $2^{100} - 1 \approx 10^{30}$ frequent sequences. These sequences are $<a_1>$, ..., $<a_{100}>$, $<a_1\ a_2>$, ..., $<a_{99}\ a_{100}>$, ..., $<a_1\ a_2\ \ldots\ a_{100}>$. For same minimum support S has only one Max sequence, $<a_1\ a_2\ \ldots\ a_{100}>$, and two Closed sequences, $<a_1\ a_2\ \ldots\ a_{50}>$, $<a_1\ a_2\ \ldots\ a_{100}>$. This example shows that if we choose $\mathcal{F}$ as our candidate set we have to consider around 10^{30} sequences in order to find Max, or Closed, sequences.

MaxSequence, and ClosedSequence use prefix projection to mine frequent sequences in a sequence database. Considering the properties of Max, and Closed sequences, and how prefix projection works, these programs mine only a subset of frequent sequences, and later they use only a subset of mined sequences as their candidate sets. It can be proved that these candidate sets are supersets of the complete set of frequent Max, and Closed, sequences. In next two sections we will see how these programs use smaller subsets of $\mathcal{F}$ as their candidate sets. In 5.1.1 a method called *Common Prefix Detection* is introduced. Using this method in prefix projection, MaxSequence, and ClosedSequence will mine a subset of frequent sequences that are likely to be Max, or Closed, sequences. How MaxSequence, and ClosedSequence do select the candidate sequences from the mined sequences is described in section 5.1.2.

5.1.1. Common Prefix Detection

Like PrefixSpan MaxSequence, and ClosedSequence use prefix projection to mine frequent sequences, but unlike PrefixSpan they do not mine the complete set of frequent sequences. For a projected database instead of doing the projection for every frequent item, MaxSequence, and ClosedSequence look for a common prefix in the database, and if they find one they will do the projection based on that prefix. If they do not find a common prefix, then like PrefixSpan, for each frequent item they will create a projected database. This way these two programs will generate only a subset of the complete set of frequent sequences, $\mathcal{F}$. Lets call this subset $\mathcal{P}$, and the complete set of Max, and Closed

sequences as $\mathcal{M}$, and C respectively. In this section first I will illustrate this method using an example, and later I will prove that $\mathcal{M}$, and C are subsets of $\mathcal{P}$.

Sequence_id	Sequence
10	$< a_1\ a_2\ \ldots\ a_{10}>$
20	$< a_2\ \ldots\ a_{10}>$
30	$< a_2>$

Table 5.2 Sequence database $\mathcal{S}$.

Lets consider another extreme case, the sequence database $\mathcal{S}$ given in Table 5.2, and the minimum support of 1 (i.e., every occurrence is frequent). PrefixSpan first scans $\mathcal{S}$ to find the frequent items. In this case all items, $a_1 \ \ldots \ a_{10}$, are frequent. These frequent items are the length-1 frequent sequences. Then for each frequent item, it creates item's corresponding projected database. For each projected database these steps are repeated, until the database has no frequent items. In each repetition the length of the prefixes increase by one, and the prefixes are added to the set of frequent sequences. Table 5.3 shows the projected databases created for $\mathcal{S}$ during this process. For sequence database $\mathcal{S}$, 511 non-empty projected databases, and 1023 frequent sequences will be generated.

Prefix	Support	Projected database
$< a_1>$	1	$< a_2\ \ldots\ a_{10}>$
$< a_2>$	3	$< a_3\ \ldots\ a_{10}>, < a_3\ \ldots\ a_{10}>, < >$
$< a_3>$	2	$< a_4\ \ldots\ a_{10}>, < a_4\ \ldots\ a_{10}>$
⋮	⋮	⋮
$< a_{10}>$	2	$< >, < >$
$< a_1\ a_2>$	1	$< a_3\ \ldots\ a_{10}>$
⋮	⋮	⋮
$< a_9\ a_{10}>$	2	$< >, < >$
⋮	⋮	⋮
$< a_1\ a_2\ \ldots\ a_{10}>$	1	$< >$

Table 5.3 Projected databases for sequence database $\mathcal{S}$.

Now lets see how MaxSequence, and ClosedSequence use prefix detection to generate less number of frequent sequences. They follow the same steps as PrefixSpan. The only difference is after finding frequent items for a database, they check to see if they can find a common prefix among the non-empty sequences in the database. If there exists a common prefix, the projection continues based on that prefix, instead of all frequent items. If there does not exist a common prefix, the projection continues based on the frequent items. In case of this example, there is no common prefix in S, so the first iteration will be like PrefixSpan, and projected databases for a_1 to a_{10} are created. The $<a_1>$–projected database contains only one sequence $<a_2 \ldots a_{10}>$. For this projected database there exists the common prefix of $<a_2 \ldots a_{10}>$. So in next iteration the projection will be done only based on $<a_2 \ldots a_{10}>$, instead of $<a_2>, <a_3>, \ldots, <a_{10}>$. The $<a_2>$–projected database contains two non-empty sequences $<a_3 \ldots a_{10}>$, and $<a_3 \ldots a_{10}>$, and the common prefix for them is $<a_3 \ldots a_{10}>$, so the next projection will be based on $<a_3 \ldots a_{10}>$. Table 5.4 shows the projected databases created for S during this process. The underlined items in the table are the detected common prefixes. For given sequence database, and minimum support, MaxSequence, and ClosedSequence will generate 9 non-empty projected databases, and 19 frequent sequences.

Prefix	Support	Projected database
$<a_1>$	1	$<\underline{a_2 \ldots a_{10}}>$
$<a_2>$	3	$<\underline{a_3 \ldots a_{10}}>, <\underline{a_3 \ldots a_{10}}>, <>$
$<a_3>$	2	$<\underline{a_4 \ldots a_{10}}>, <\underline{a_4 \ldots a_{10}}>$
⋮	⋮	⋮
$<a_{10}>$	2	$<>, <>$
$<a_1 \ldots a_{10}>$	1	$<>$
$<a_2 \ldots a_{10}>$	2	$<>, <>$
$<a_3 \ldots a_{10}>$	2	$<>, <>$
⋮	⋮	⋮
$<a_9\ a_{10}>$	2	$<>, <>$

Table 5.4 Projected databases for S, using prefix detection.

Now we need to show that the set of frequent sequences generated using this method, $\mathcal{P}$, covers the set of Max, and Closed sequences. In other word we need to prove that:

$$\mathcal{M} \subseteq C \subseteq \mathcal{P}.$$

Assume a projected database, α-projected (α is a sequence), with n non-empty sequences in it. For any given frequent sequence, ϕ, in α-projection it is easy to see that $support_{\alpha\_projected}(\phi) \leq n$. This means the maximum support for a frequent sequence obtained from a projected database, is equal to the number of non-empty sequences in that database. So if we can find a common prefix, β, among all the non-empty sequences in α-projected database, then sequence β will have the maximum possible support in the database. For any given sequence γ, such that γ is a subsequence of β, we will have the following:

$$support_{\alpha\text{-}projected}(\beta) = n,$$
$$support_{\alpha\text{-}projected}(\gamma) \leq n,$$
$$support_{\alpha\text{-}projected}(\beta) \leq support_{\alpha\text{-}projected}(\gamma).$$

From these relations we can conclude that :

$$support_{\alpha\text{-}projected}(\gamma) = support_{\alpha\text{-}projected}(\beta) = n.$$

From this conclusion, and the definitions of Max, and Closed sequence, we can see that sequence γ cannot be a Max, or Closed, sequence, because sequence β, a proper supersequence of γ, is also frequent, and has the same support as γ. This reasoning is true for the original sequence database $\mathcal{S}$, that α-projected is projected from that. This means that sequence $\alpha\beta$ will be the supersequence of the sequence $\alpha\gamma$, and $support_S(\alpha\gamma) = support_S(\alpha\beta)$, and from these we can conclude that the sequence $\alpha\gamma$ cannot be a Max, or Closed sequence. Also when we create the $\alpha\beta$–projected database from the

α–projected database, any frequent item in α–projected database that is not covered in the common prefix, β, will remain frequent in the resulting αβ–projected database, and resulting frequent sequences from these items will be mined in the later iterations of the process. This shows that during the mining process using common prefix detection method, we are only ignoring frequent sequences that do not have the potential to be Max, or Closed sequences, and therefore we can say that $\mathcal{P}$ is the superset of $\mathcal{M}$, and C.

The difference between the number of frequent sequences obtained using PrefixSpan with, and without common prefix detection is significant for the extreme case we studied in this section, but what about more general non extreme cases? In general cases, and specially for high values of support threshold, there might not be a significant advantage in using common prefix detection, and the overhead of this detection might even make the mining process slower. But for dense sequence databases, and for low values of the support threshold, this method will show its advantages. In general as the projection process goes further, and the resulting projected databases tend to have less number of sequences, it is more probable to detect a common prefix in the database. As a result in general cases, it seems to be useful to use the common prefix detection, only when number of the sequences in a projected database, falls below a certain limit. Further studies can be done to find a proper value for this limit.

5.1.2. Candidate Selection

In previous section I showed how we can mine a subset of frequent sequences, $\mathcal{P}$, that covers all the Max, and Closed sequences. In this section we will see how we can choose subsets of $\mathcal{P}$, as candidate sets for Max, and Closed sequence mining.

Like PrefixSpan, MaxSequence, and ClosedSequence programs use recursion to mine frequent sequences. In a given database they find the frequent items (or a common prefix), and create the corresponding projected databases for those frequent items (or that prefix), and then recursively do the same for each projected database until the resulting

projected database contains no frequent items, which in that time the recursion for that branch ends, and a backtrack (returning to the calling procedure at the end of the recursion branch) happens. In each iteration of this process the prefix of the projected database is added to the set of frequent sequences. Also in each iteration the prefixes of the projected databases are created by growing the prefix of the original database by appending either a frequent item, or a common prefix to end of it. This means if during the mining process β–projected database is created from α–projected database, then we can say that $\alpha \subset \beta$. Starting from the original database, and moving down in the recursion tree the size of the prefix grows until we cannot go any further (a backtrack happens). This means that if β–projected database contains no frequent items, then β is the supersequence of all the sequences before it. From this fact, and the definition of the Max sequence we can see that none of the sequences before β can be a Max sequence, and only β has the potential to be a Max sequence. This suggests that for mining Max sequences, it is sufficient to use only the sequences generated during backtracks as our candidate set. MaxSequence uses these sequences as its candidate set.

Figure 5.1 illustrates this process for sequence database S, shown as the root table in the figure, and minimum support of one. Sequences in the original database have no common prefix, so the projection happens based on the frequent items. There are five frequent items, so five projected databases, one for each frequent item, are generated. The $<a_1>$–Projected database has one sequence, so the projection happens based on that sequence, and $<a_1 a_2\ a_3\ a_4\ a_5>$–Projected database, with no sequences in it, is created. Since this projected database is empty, a backtrack happens, and the process returns to the higher level in the tree (that is the $<a_2>$–Projected database). The same thing happens for $<a_2>$, $<a_3>$, and $<a_4>$ projected databases. They all have common prefixes, and projection continues based on those common prefixes. $<a_5>$–Projected database is empty, and no further projection is necessary. The prefix of each projected table is a frequent sequence, and candidate sequences will be chosen from these frequent sequences. As it was described earlier among frequent sequences generated, only those that are generated during a backtrack are selected as the candidate sequences. In this case

the candidate sequences are $<a_1\ a_2\ a_3\ a_4\ a_5>$, $<a_2\ a_3\ a_4\ a_5>$, $<a_3\ a_4\ a_5>$, $<a_4\ a_5>$, $<a_5>$.

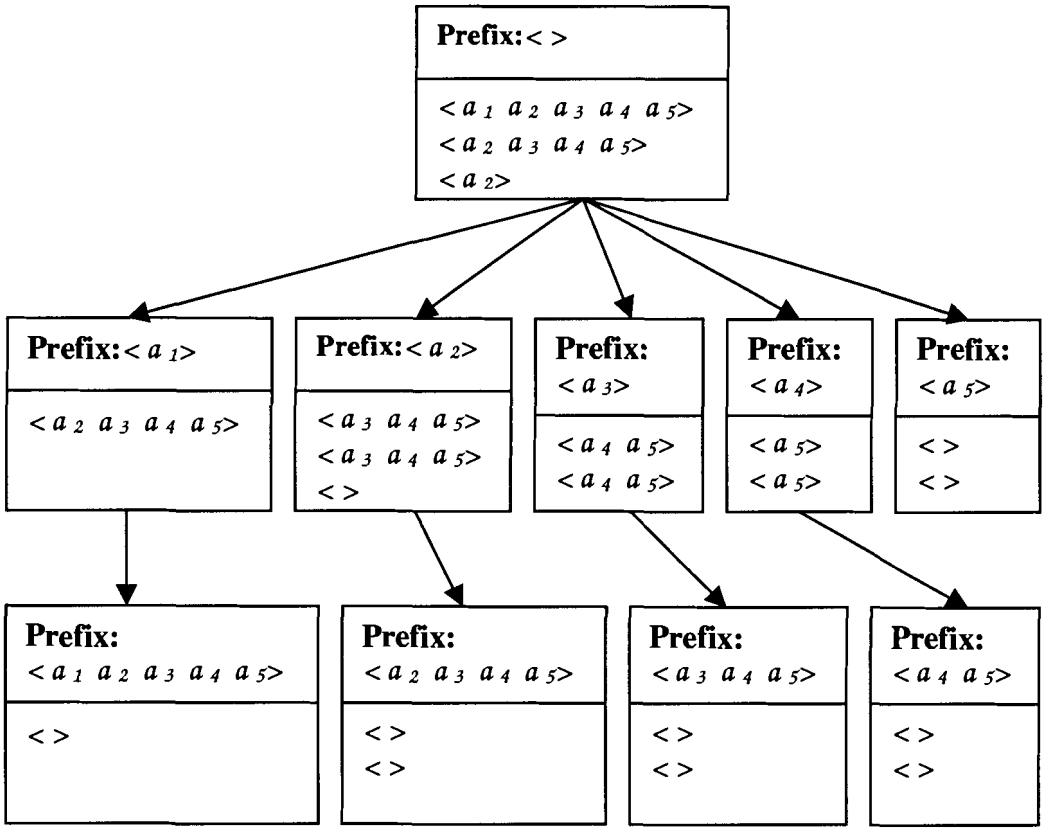

Figure 5.1 Example for MaxSequence candidate generation, and selection.

The same candidate set cannot be used for Closed sequence mining, because although the prefix of the projected database is the supersequence of the prefix of the original database, but their support are not necessarily equal. Given the sequence database S, as during the mining process β–projected database is created from α–projected database, the number of sequences in the β–projected database will be less than or equal to the number of the sequences in the α–projected database. This suggests that:

$$Supports_S(\beta) \leq Supports_S(\alpha).$$

Lets consider the following two cases:

1. $Supports(\beta) = Supports(\alpha)$: In this case α cannot be a Closed sequence, because there exists sequence β that is a supersequence of α, and its support is not less than α's support.

2. $Supports(\beta) < Supports(\alpha)$: In this case α can be a Closed sequence, because in its recursion branch it has a bigger support than its supersequences.

From this, and the facts we used for generating candidate set for Max sequence mining, we can generate the candidate set for Closed sequence mining in the following way. We add the sequence generated during a backtrack to the candidate set. Going up on the recursion branch we ignore the sequences that have the same support as the last sequence added to the candidate set, and only add sequences that have higher support than the last sequence added to the candidate set.

Implementing the mining procedure, in a depth first fashion is not efficient. In mining process for each projected database we need various data structures to keep track of frequent items, and some other information. These data structures can be big, and storing them in stack, for each iteration of the program, might not be practical for big databases. Therefore some of these structures are defined globally. For each database after finding its frequent items by scanning it once, we create all the corresponding projected databases for it at the same iteration. After doing this we no longer need the information inside global structures, and next iteration can overwrite this information. As described earlier for generating the candidate set for ClosedSequence we need to move up the recursion track as we encounter a backtrack, but since we do not keep the frequent item information for previous iterations, we need to keep track of sequences as they are generated. For doing this a tree like data structure is used. As projected databases are created from an original database, the corresponding frequent sequences generated from these projected databases are added to the tree as the children of the sequence generated from the original

database. A sequence generated from a database that does not have any frequent items, represents a leaf node in the tree. Closed sequence uses this structure to generate its candidate set. As it reaches a leaf node, it adds this node to the candidate set, and moves to the parent node, if parent node has higher support, then it will be added to the candidate set, otherwise the upper node in the tree will be checked. The process will repeat these steps until it reaches the root node. All the childless nodes can be deleted from the tree, after they have been check.

I will illustrate the process of candidate generation for ClosedSequence using the example sequence database shown in figure 5.1. Same process, as it described for MaxSequence, will happen for ClosedSequence frequent sequence generation, but the selection process will be different. Figure 5.2 shows the recursion tree for this process. The numbers following the sequences in each node is the support for that sequence. Below the root we have all the frequent items for the main sequence database. The child parent relation shows from which parent databases the child databases are projected.

For ClosedSequence candidate selection, lets start by checking the left most branch. As soon as we get to the leaf node, we will add it to the candidate set, and since the parent node does not have a higher support we will ignore it. For the next branch we will add the leaf node, and since its parent node has a higher support we will also add the parent to the candidate set. For other branches in this case we only add the leaf nodes to the candidate set. The candidate set in this case will contain these sequences $< a_1 \ldots a_5 > : 1$, $< a_2 \ldots a_5 > : 2$, $< a_2 > : 3$, $< a_3 \ldots a_5 > : 2$, and $< a_4 \ a_5 > : 2$.

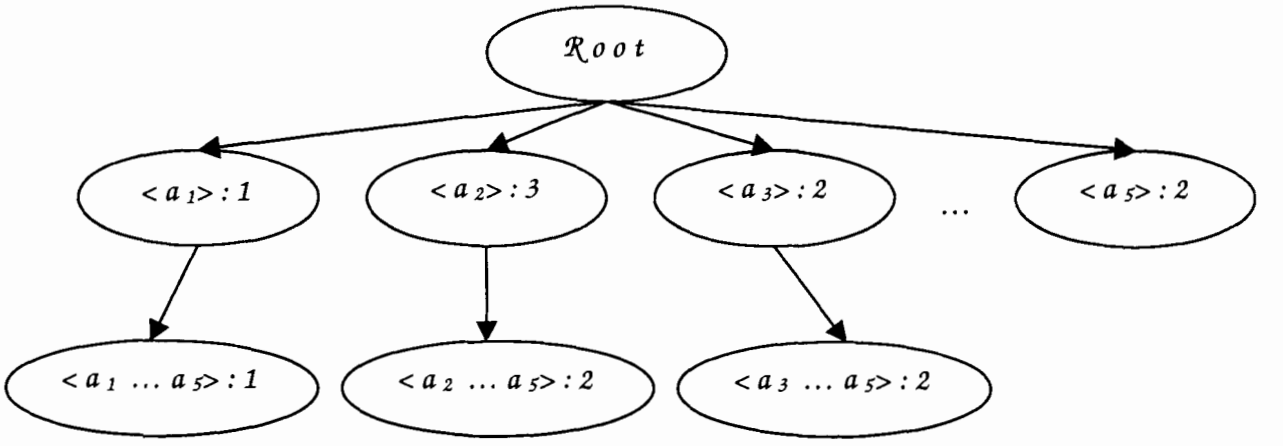

Figure 5.2 A sample Recursion tree.

Algorithms for MaxSequence, and ClosedSequence are shown in algorithm 5.2, and 5.3. How to check a candid sequence, and to add it to the result set if it is a Max, or Closed sequence is shown in algorithm 5.4. Routines used in this algorithm will be discussed in the next chapter. There will be two sets of routines, one set for Max sequence mining, and one set for Closed sequence mining. Recursion tree routines that are used in ClosedSequence are shown in Algorithm 5.1.

Subroutine AddToRecursionTree(RecursionTree, aSequence, aParent)
Parameters: RecursionTree: a tree structure to keep track of the order of sequence generation.; aSequence: is a sequence; apparent: is a sequence.
Method:
Add aSequence to the RecursionTree as a child of aParent

Subroutine EmptyRecursionTree(RecursionTree, aSequence, aTree)
Parameters: RecursionTree: a tree structure to keep track of the order of sequence generation.; aSequence: is a sequence; aTree: is a Max-Tree, or Closed-Tree for storing sequences;
Method:

```
LastSupport = 0
For each sequence aSeq, starting from aSequence to all of its ancestors
    If aSeq has no children
        If aSeq has a support higher than LastSupport
            LastSupport = Support of aSeq
            Call AddToResultset( aTree, aSeq )
        Delete aSeq from RecursionTree
    Else
        Exit for loop.

For rest of the remaining ancestors

    If the support of the ancestor is less than or equal the LastSupport

        Set the support of the ancestor to –1. // No need to consider this sequence for
        rest of its children.
```

Algorithm 5.1 Recursion tree routines.

Input: A sequence database S, and minimum support threshold ξ.
Output: The complete set of Max sequential patterns.
Method:

MaxTree = { }.
Call **MaxSequence**(<>, 0, S, MaxTree)
Output MaxTree.

Subroutine MaxSequence(α, ℓ, $S|_\alpha$, MaxTree)
Parameters: α: a sequence; ℓ: the length of α; $S|_\alpha$: the α-projected database, if $\alpha \neq <>$, otherwise the sequence database S; MaxTree: a Max-Tree for storing results.
Method:

Scan $S|_\alpha$ once, find the set of frequent items b such that:
b can be assembled to the last element of α to form a sequence;
or $<b>$ can be appended to α to form a sequence.

If the set of frequent items for $S|_\alpha$ is empty
Call **AddToResultset**(MaxTree, α)
Return

Try to find a common prefix β among all non-empty sequences in $S|_\alpha$ such that:
Firs element of β can be assembled to the last element of α to form a sequence;
or β can be appended to α to form a sequence.

If β is found
Append β to α to from a sequence α',
Construct α'-projected database $S|_{\alpha'}$,
Call **MaxSequence** (α', l + length(β), $S|_{\alpha'}$, MaxTree).
Else
For each frequent item b,
Append it to α to form a sequence α',
Construct α'-projected database $S|_{\alpha'}$,
Call **MaxSequence** (α', l + 1, $S|_{\alpha'}$, MaxTree).

Algorithm 5.2 MaxSequence a program for Max sequence mining.

Input: A sequence database $\mathcal{S}$, and minimum support threshold ξ.
Output: The complete set of Max sequential patterns.
Method:

ClosedTree = { }, RecursionTree = { }.
Call **ClosedSequence**(<>, 0, $\mathcal{S}$, ClosedTree, RecursionTree)
Output ClosedTree.

Subroutine ClosedSequence(α, ℓ, $\mathcal{S}|_\alpha$, ClosedTree, RecursionTree)
Parameters: α: a sequence; ℓ: the length of α; $\mathcal{S}|_\alpha$: the α-projected database, if $\alpha \neq$ <>, otherwise the sequence database $\mathcal{S}$; ClosedTree: a Closed-Tree for storing results; RecursionTree: a tree structure to keep track of the order of sequence generation.
Method:

Scan $\mathcal{S}|_\alpha$ once, find the set of frequent items b such that:
b can be assembled to the last element of α to form a sequence;
or $<b>$ can be appended to α to form a sequence.

If the set of frequent items for $\mathcal{S}|_\alpha$ is empty
Call **EmptyRecursionTree**(RecursionTree, α)
Return

Try to find a common prefix β among all non-empty sequences in $\mathcal{S}|_\alpha$ such that:
Firs element of β can be assembled to the last element of α to form a sequence;
or β can be appended to α to form a sequence.

If β is found
Append β to α to from a sequence α',
Construct α'-projected database $\mathcal{S}|_{\alpha'}$,
Call **AddToRecursionTree**(RecursionTree, α', α),
Call **ClosedSequence** (α', l + length(β), $\mathcal{S}|_{\alpha'}$, ClosedTree, RecursionTree).
Else
For each frequent item b,
Append it to α to form a sequence α'.
Construct α'-projected database $\mathcal{S}|_{\alpha'}$,
Call **AddToRecursionTree**(RecursionTree, α' , α),
Call **ClosedSequence** (α', l + 1, $\mathcal{S}|_{\alpha'}$, ClosedTree, RecursionTree).

Algorithm 5.3 ClosedSequence a program for Closed sequence mining.

```
Subroutine AddToResultset( aTree, aSequence )
Parameters: aTree: is a Max-Tree, or Closed-Tree for storing sequences; aSequence: is a
    sequence.
Method:
    If SuperSequenceExists( aTree, aSequence )
        Return

    DeleteSubSequences( aTree, aSequence )

    Add aSequence to aTree
```

Algorithm 5.4 Routine to add a sequence to a resultset tree.

5.2. Checking Candidates

In previous section I described how MaxSequence, and ClosedSequence programs generate their candidate sets. In this section I will describe how to find the Max, and Closed, sequences from the candidate sequences.

MaxSequence, and ClosedSequence, keeps a list of sequences as their result sets. At the beginning these sets are empty. As a candidate sequence, α, is generated, it is checked with the sequences inside the result set. If a Max, or Closed, supersequence of α is found inside the result set, α will be ignored, otherwise all the Max, or Closed, subsequences of α will be deleted from the result set, and α will be added to the set. After checking all the candidate sequences, the result set will contain the complete set of Max, or Closed, sequences. Given a sequence α, and a result set, the performance of this check depends on how fast we can answer these two queries. First if there exists a Max, or Closed, supersequence of α in the result set, and second which sequences in the result set are Max, or Closed, subsequences of α. MaxSequence, and ClosedSequence programs use two special data structures, *Max-Tree*, and *Closed-Tree*, as their result sets. These data structures are designed in a way that can answer the checking queries efficiently.

These data structures are similar to *CR-Tree* [3], and are described in the next two sections. Section 5.2.1 describes Max-Tree, the result set for MaxSequence, and section 5.2.2 talks about Closed-Tree, the result set for ClosedSequence.

5.2.1. Max-Tree

Max-Tree is a CR-Tree like structure that is used as the result set for MaxSequence program. In this section we will discuss the structure of the Max-Tree, and we will also see how it facilitates the candidate checking in the Max sequence mining.

This structure has two major parts, a *tree structure*, and an array of linked lists called *node link table*. Each node in the tree represents an element, and each path (starting from the root, and ending in a leaf) in the tree represents a sequential pattern. Sequences with a common prefix will share the element nodes of their common prefix. This means that the sequences are stored in the tree in compressed form.

In the node link table, a linked list is kept for each frequent item in the sequence database. A node in these lists is a pointer to a tree node. A tree node will be in the linked list of the item a, if it is the last element of a sequence that contains a. These lists are kept sorted based on the level of the tree node they are pointing at. The level of an element in the tree in fact shows the ordinal number of that element in the sequence.

Figure 5.3 shows a sample Max-Tree. Each node in the tree represents an element, and each path from the root to a leaf node represents a sequence. This tree has three leaves, so it represents three sequences. These sequences are $<i_1 i_3 i_2 (i_3 i_4)>$, $<i_1 i_5>$, and $<i_3 i_1 i_4>$. Dashed curve lines represent the linked lists for each item in the node link table. Notice that the node for element i_1 in the right most branch has a higher tree level, than the element in the left most branch, so it will be the first item in the linked list for item i_1.

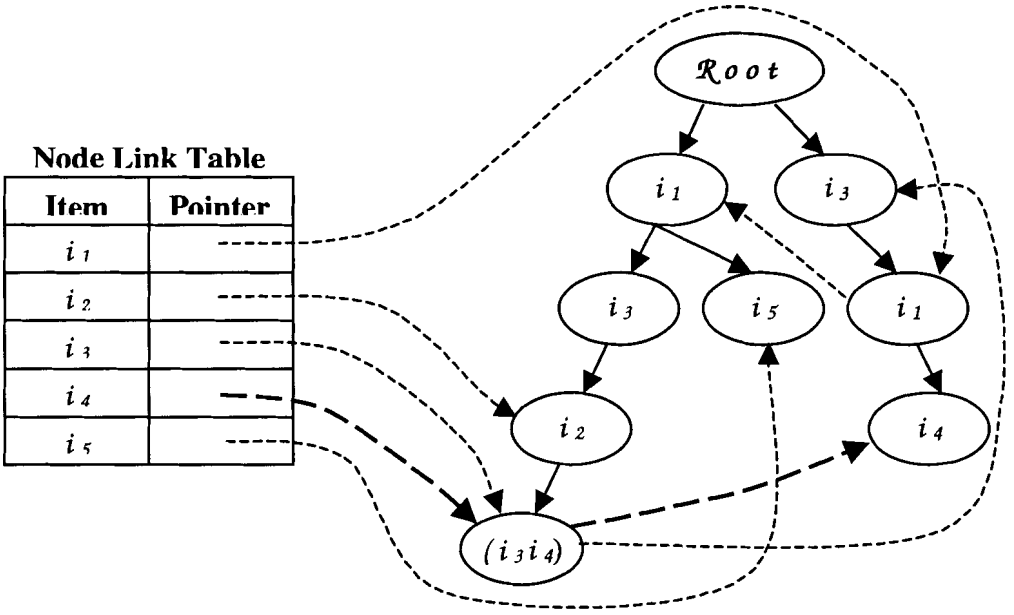

Figure 5.3 A sample Max-Tree.

Given a sequence α, lets see how MaxSequence, using Max-Tree, answers this query that if result set contains a Max supersequence of α. Assume that the last element of α is s_n, and it contains these items $(i_1 i_2 \ldots i_m)$. From the items in s_n the item, i_x where $1 \leq x \leq m$, with the shortest linked list, $\mathcal{L} i_x$, in the node link table is chosen. Lets call the node in the tree pointed to by the first node in $\mathcal{L} i_x$ the current node. The sequence starting from the root, and ending with the current node is checked to see if it is a Max supersequence of α. If α is contained by this sequence then the answer to the query is yes, otherwise the next item in $\mathcal{L} i_x$ will be chosen as the current node, and the containment check will be repeated. This will continue until either the last node of $\mathcal{L} i_x$ is checked, or the level of the current node becomes less than the length of α (a shorter sequence cannot contain a longer one). Since nodes in $\mathcal{L} i_x$ are sorted based on the tree level, the process can be stopped as soon as a node with lesser level value than the length of α is encountered. In this case the answer to the query will be no.

Lets check this process on the tree provided in figure 5.3. Assume sequence α = $< i_1 (i_3 i_4) >$. Last element of α is $(i_3 i_4)$. The linked lists for i_3, and i_4 are both two nodes long, so we pick one randomly. Here we choose the linked list for item i_3. The first node this linked list points to represents the sequence $< i_1 i_3 i_2 (i_3 i_4) >$ (starting from the root, and ending with this node). This sequence is a supersequence of α, so the process stops, and the answer to the query is yes. For sequence $\alpha = < i_1 i_3 i_5 >$, we pick the linked list of the item i_5. This link list only contains one node, and its level is two. Since length of α is three, the sequence in the tree cannot be a supersequence of α. The process stops, and the answer to the query is no.

To delete the sequences in the Max-Tree that are Max subsequences of a given sequence α program performs the following steps. For each item i, in α the following steps will be performed. Lets call the linked list of item i in the node link table $\mathcal{L}i$. Starting from the last node in $\mathcal{L}i$ for every node in $\mathcal{L}i$ that points to a leaf node in the tree, and its corresponding tree node has a smaller, or equal level than the length of α, check if the sequence starting from the root, and ending by this node is a Max subsequence of α. If yes delete the sequence from the tree, and continue. Again since nodes in $\mathcal{L}i$ are sorted based on the tree level, the process for this linked list can be stopped as soon as a node with bigger tree level value than the length of α is encountered.

Again lets use the tree shown in figure 5.3 to illustrate this process. Assume the sequence $\alpha = < i_1 i_3 i_5 >$. We have to check all the nodes in the linked lists of all the items in the α. Lets start with i_1. Linked list for i_1 has only one node, and since this node does not point to a leaf node, we ignore it. Starting from the last node in the i_3 linked list, again this node does not point to a leaf, so it is ignored. The next node in this linked list (first node in the linked list) points to a leaf node that represents the sequence $< i_1 i_3 i_2 (i_3 i_4) >$. The level of this node is four, but the length of α is three, so it is ignored. The next linked list to check is the linked list for item i_5. This linked list contains one node that points to a leaf. This leaf node represents the sequence $< i_1 i_5 >$ in

the tree, and since it is a subsequence of α, it should be deleted from the tree. The tree after deletion of this sequence is shown in figure 5.4.

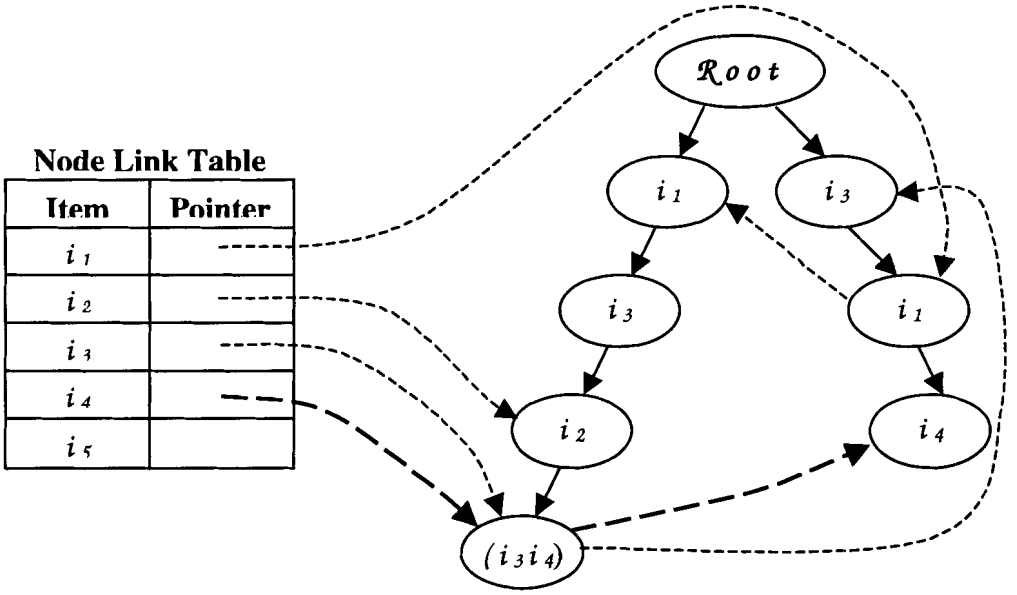

Figure 5.4 Max-Tree after deletion.

To check a sequence in the tree (specified by its last node in the tree) with a given sequence α, we compare the last element of α with the last tree node, and move toward the first element of α and the parent of the compared tree node. The tree node level, and length of α can also help in the checking process. To delete a sequence from the tree, the program starts from the last element of the sequence, the leaf node, toward the first element of the sequence, the root node, and deletes every node that does not have any other children. The corresponding pointers in the node link table should also be deleted.

Routines that are used for checking a candid sequence, and adding it to a Max-Tree, if it is a Max sequence are shown in algorithm 5.5. These are the routines that can be used in algorithm 5.4 in previous section, for Max sequence mining.

// Returns true if a proper Max supersequence of aSequence exists in aTree.
Boolean Function SuperSequenceExists(aTree, aSequence)
Parameters: aTree: is a Max-Tree for storing sequences; aSequence: is a sequence.
Method:
i is the item in the last element of aSequence with shortest link list in aTree's node link table.

For every pointer in the *i* 's linked list, starting from the beginning of the list
aNode is the tree node that this pointer pointing to.
If level of aNode is less than the length of aSequence
Exit for loop

TreeSequence is the sequence represented in aTree by the path starting from the root to aNode.

If aSequence is the Max subsequence TreeSequence
Return True

Return False

Subroutine DeleteSubSequences (aTree, aSequence)
Parameters: aTree: is a Max-Tree for storing sequences; aSequence: is a sequence.
Method:
For each element, anElement, in aSequence

For each item, anItem, in anElement

For every pointer in the anItem's linked list in aTree's node link table that points to a leaf node, starting from the end of the linked list.

aNode is the tree node that this pointer pointing to.

If level of aNode is greater than the length of aSequence
Exit inner for loop

TreeSequence is the sequence represented in aTree by the path starting from the root to aNode.

If aSequence is the Max supersequence TreeSequence
Delete TreeSequence from aTree

Algorithm 5.5 Max-Tree routines.

5.2.2. Closed-Tree

Closed-Tree is also a CR-Tree like structure that is used as the result set for ClosedSequence program. In this section we will discuss the structure of the Closed-Tree, and we will also see how it facilitates the candidate checking in the Closed sequence mining.

Like Max-Tree, this structure has two major parts, a *tree structure*, and an array of linked lists called *node link table*. But unlike Max-Tree each node in the tree represents an item. Nodes can be part of different sequences, and among the different support values for those sequences each node stores the highest support value. Each node also has a flag that shows if it is an *intra* item. Non-first items in a multiple item element are intra items. A node with higher support than all of its children is called a *terminal node*. A path from root to a terminal node represents a sequence in the tree. Sequences with a common prefix will share the item nodes of their common prefix. This means that the sequences are stored in the tree in compressed form.

Like Max-Tree, a linked list is kept for each frequent item in the node link table. A node in these lists is a pointer to a tree node. A tree node will be in the linked list of the item a, if it is the last instance of item a in a sequence.

Figure 5.5 shows a sample Closed-Tree. Each node in the tree represents an item, and each path from the root to a terminal node represents a sequence. The underlined item is an intra node. This tree has four terminal nodes, so it represents four sequences. These sequences are $<i_1 i_2 i_1>:2$, $<(i_1 i_3) i_3>:3$, $<i_2 i_3>:3$, and $<i_2 i_3 i_1>:1$. Dashed curve lines represent the linked lists for each item in the node link table.

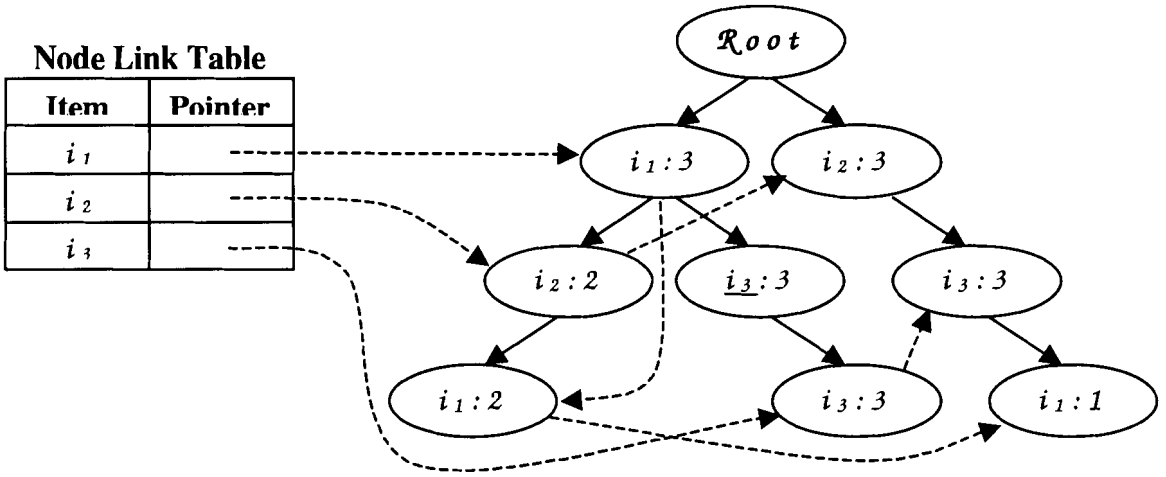

Figure 5.5 A sample Closed-Tree.

Given a sequence α, lets see how ClosedSequence, using Closed-Tree, answers this query that if result set contains a Closed supersequence of α. Lets call the last item in α, i_x, and the link list for this item in the node link table, $\mathcal{L}i_x$. Sequence α is checked with all the sequences that start from the root, and end with a node pointed to by an element in $\mathcal{L}i_x$. If a Closed supersequence of α is found the process stops, and the answer the query is yes, otherwise answer is no.

Lets examine this process on the tree shown in figure 5.5. Assume $\alpha = <i_2 i_3>:2$. The linked list for i_3, the last item of α, has two nodes. The first node represents the sequence $<(i_1 i_3)i_3>:3$. This sequence is not a supersequence of α, so we move to the second node in the linked list. This node represents the sequence $<i_2 i_3>:3$, and this sequence is a closed supersequence of α, so the process stops, and answer to the query is yes. The query for sequence $\alpha = <i_2 i_3>:4$ will follow the same steps, but since the sequence $<i_2 i_3>:3$ is not a Closed supersequence of α, and there are no more nodes in the linked list of item i_3, process stops, and the answer to the query will be no.

To delete the Closed subsequences of a given sequence α, in the Closed-Tree the following steps are performed. For each item i, in α the following steps will be performed. Lets call the linked list of item i in the node link table $\mathcal{L}i$. For all the nodes in $\mathcal{L}i$ that point to a terminal node in the tree, and their corresponding tree node has a smaller or equal support, and level than the support, and length of α, check if the sequence starting from the root, and ending by this node is a Closed subsequence of α. If yes delete the sequence from the tree, and continue.

Lets examine sequence $\alpha = <i_1> : 4$ on the tree shown in figure 5.5, and see if there are Closed subsequences of α in the tree. The linked list for item i_1 contains three nodes. The one that represents the sequence $<i_1> : 3$, is not terminal, so it is ignored. The second and third nodes, that represent sequences $<i_1 i_2 i_1> : 2$ and $<i_2 i_3 i_1> : 1$ respectively , are terminal, but their length is more than α, so they cannot be α's Closed supersequences. Since there are no more nodes in the linked list to check, the answer to the query is no. For $\alpha = <i_2 i_3> : 4$, we start with the link list of the item i_2. This linked list points to two nodes, but none of them are terminal nodes, so we move to the linked list of i_3. This linked list points to two terminal nodes in the tree. The one that represents the sequence $<(i_1 i_3) i_3> : 3$ has higher level than the length of α, so it cannot be a subsequence of α. The one that represents the sequence $<i_2 i_3> : 3$ has less support , and level, than the support, and the length of α, so it is checked with α, and since it is a Close subsequence of α, it is deleted from the tree. The tree after deleting the sequence $<i_2 i_3> : 3$ from it, is shown in figure 5.6.

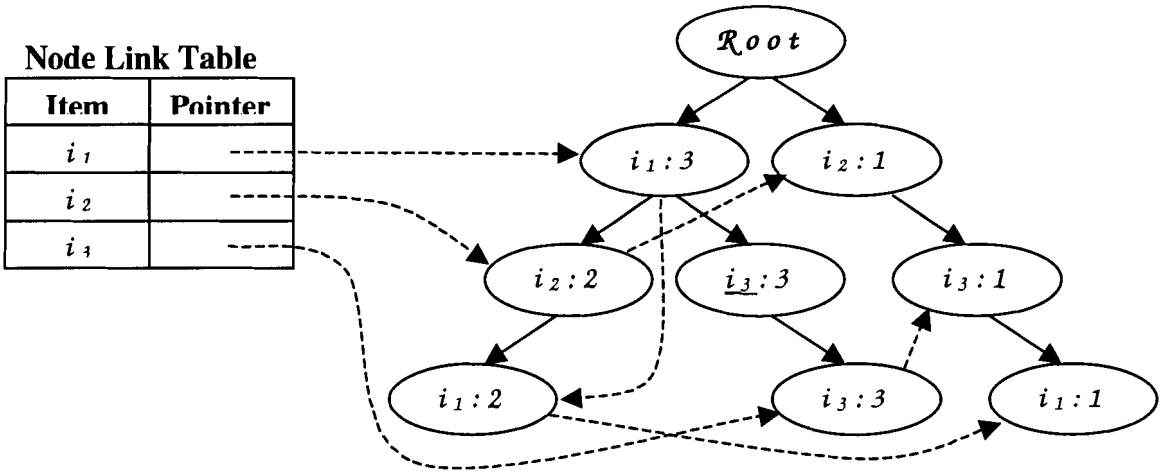

Figure 5.6 Closed-Tree after deletion.

To check a sequence in the tree (specified by its last node in the tree) with a given sequence α, we compare the last item of α with the last tree node, and move toward the first item of α and the parent of the compared tree node. Comparing tree node level, and support with length, and support of α can also speed up the process. To delete a sequence from the tree, the program starts from the last element of the sequence, the terminal node, toward the first element of the sequence, the root node, and deletes every node that does not have any other children. The support of the nodes that are not deleted from the tree (nodes that have children) should be updated, and the corresponding pointers in the node link table should be deleted.

Routines that are used for checking a candid sequence, and adding it to a Closed-Tree, if it is a Closed sequence are shown in algorithm 5.6. These are the routines that can be used in algorithm 5.4 in previous section, for Closed sequence mining.

// Returns true if a proper Closed supersequence of aSequence exists in aTree.
Boolean Function SuperSequenceExists(aTree, aSequence)
Parameters: aTree: is a Closed-Tree for storing sequences; aSequence: is a sequence.
Method:
    i is the last item of aSequence.

    For every pointer in the i's linked list
        aNode is the tree node that this pointer pointing to.
        If level of aNode is not less than the length of aSequence
            TreeSequence is the sequence represented in aTree by the path starting from the root to aNode.

            If aSequence is the Closed subsequence TreeSequence
                Return True

    Return False

Subroutine DeleteSubSequences (aTree, aSequence)
Parameters: aTree: is a Closed-Tree for storing sequences; aSequence: is a sequence.
Method:
    For each element, anElement, in aSequence

        For each item, anItem, in anElement

            For every pointer in the anItem's linked list in aTree's node link table that points to a terminal node, starting from the end of the linked list.

                aNode is the tree node that this pointer pointing to.

                If level of aNode is not greater than the length of aSequence

                    TreeSequence is the sequence represented in aTree by the path starting from the root to aNode.

                    If aSequence is the Closed supersequence TreeSequence
                        Delete TreeSequence from aTree

Algorithm 5.6 Closed-Tree routines.

5.3. *String Elimination*

String Elimination is another method that can be used to further improve the performance of Max sequence mining. As we create the projected databases we can check the sequences in the database with the result set. If a sequence in a projected database is a subsequence of a sequence in the result set, then we can delete that sequence from the projected database. This will work for Max sequence mining, because we are looking for the longest frequent sequences, and we are not interested in shorter sequences, even if they have higher supports. This idea will not work for Closed sequence mining, because by eliminating sequences in projected databases, we might loss some shorter frequent sequences with higher supports.

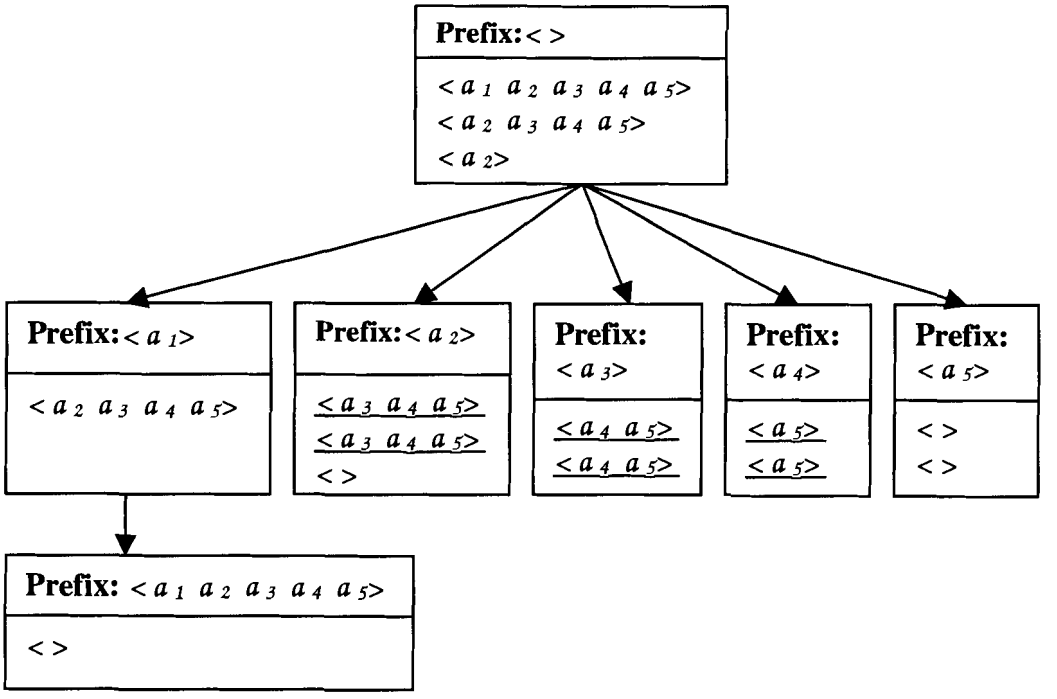

Figure 5.7 Example for String Elimination.

As an example consider the sequence database shown in figure 5.1. The result of Max, and Closed, sequence mining are $\{<a_1\ a_2\ a_3\ a_4\ a_5>\}$, and $\{<a_1\ a_2\ a_3\ a_4\ a_5>:1,\ <a_2\ a_3\ a_4\ a_5>:2,\ <a_2>:3\}$ respectively. Figure 5.7 shows the mining process on the same sequence database using string elimination. The underlined sequences in the projected databases, are the eliminated sequences. The result of Max, and Closed, sequence mining in this case are $\{<a_1\ a_2\ a_3\ a_4\ a_5>\}$, and $\{<a_1\ a_2\ a_3\ a_4\ a_5>:1,\ <a_2>:3,\ <a_3>:2,\ <a_4>:2,\ <a_5>:2\}$ respectively. As it can be seen in this case using string elimination produces the right set of sequences for Max sequence mining, but the sequences generated for Closed sequence mining are not the right ones.

Chapter 6. Results

In this chapter we compare the results, and performance of the different programs described in pervious chapters, on some sample datasets. The programs were tested using seven synthetic sequence databases, and one real database. The synthetic databases were generated using publicly available synthetic generation program of the IBM Quest data mining project [10]. This data generator has been used in many sequential pattern mining studies. Some parameters of this program are shown in table 6.1, and the descriptions of sequence databases used for testing are summarized in table 6.2. The real database, called ProductClicks, is constructed form the transactional data file, clicks.data, of the KDD Cup 2000. These Data files are described in [19]. ProductClicks shows which of the 1423 different products, and assortments, are viewed by 29369 different users, and in what order. Product pages viewed in one session are considered as an item set, and different sessions for one user is considered as a sequence.

Name	Command	Description
D	-ncust	Number of customers in 000s (default: 100)
C	-slen	Average transaction per customer (default: 10)
T	-tlen	Average items per transaction (default: 2.5)
N	-nitems	Number of different items in 000s (default: 10)
	-rept	Repetition level (default: 0)
N_S	-seq.npats	Number of sequential patterns (default: 5000)
S	-seq.patlen	Average length of maximal pattern (default: 4)
	-seq.corr	Correlation between patterns (default: 0.25)
	-seq.conf	Average confidence in a rule (default: 0.75)
N_I	-lit.npats	Number of patterns (default: 25000)
I	-lit.patlen	Average length of maximal pattern (default: 1.25)
	-lit.corr	Correlation between patterns (default: 0.25)
	-lit.conf	Average confidence in a rule (default: 0.75)

Table 6.1Command line options for IBM Quest data generator.

Experiments were performed on a 550 MHz Pentium III machine, with 256 MB of main memory. Ten programs were tested during these experiments. Summary information for these programs is shown in table 6.3.

Name	N	C	T	S	I	N_S	D	Size (MB)
C20-N100-T2.5-D100K	100	20	2.5	4	1.25	5K	100	24.9
C10-N10-T1-S4-I1.25-D100K	10	10	1	4	1.25	5K	100	5.7
C2-N10-T5-S4-I1.25-D100K	10	2	5	4	1.25	5K	100	1
C5-N10-T2.5-S4-I1.25-D100K	10	2	2.5	4	1.25	5K	100	4.4
C5-N10-T2.5-S4-I2.5-D100K	10	5	2.5	4	2.5	5K	100	4.1
C10-N10-T2.5-Ns100-D1K	10	10	2.5	4	1.25	100	1	0.12
C10-N10-T2.5-Ns5K-D1K	10	10	2.5	4	1.25	5K	1	0.12

Table 6.2 Synthetic sequence database descriptions.

In section 6.1 the performance of MaxSequence is compared with MaxNaive, and PrefixSpan. MaxSequence, and MaxNaive are described in chapters 5, and 4 respectively. In section 6.2 a similar evaluation is performed for ClosedSequence. ClosedSequence, and ClosedNaive are implementations of the algorithms described in chapters 5, and 4 respectively. The effects of the optimization methods (Common Prefix Detection, Max-Tree, Closed-Tree, and String Elimination) that are used in MaxSequence, and ClosedSequence programs are studied in section 6.3

Program	Description	Name in Charts
PrefixSpan	PrefixSpan-1 (pseudo-projection) for frequent sequence mining.	P
MaxNaive	Max sequence mining program using naïve approach described in chapter 4.	MN
MaxSequence	Max sequence mining program described in chapter 5 (without string elimination).	MS
MaxSeqSE	Max sequence mining program with string elimination.	MSSE
MaxSeqNoPre	Max sequence mining program described in chapter 5, without prefix detection.	MSNP
MaxSeq NoMaxTree	Max sequence mining program described in chapter 5, without Max-Tree (Uses a linear list like MaxNaive).	MSNMT
ClosedNaive	Closed sequence mining program using naïve approach described in chapter 4.	CN
ClosedSequence	Closed sequence mining program described in chapter 5.	CS
ClosedSeqNoPre	Closed sequence mining program described in chapter 5, without prefix detection.	CSNP
ClosedSeq NoClosedTree	Closed sequence mining program described in chapter 5, without Closed-Tree (Uses a linear list like ClosedNaive).	CSNCP

Table 6.3 Programs that are being tested.

6.1. Max Sequence Mining Results

In this section the performance of Max sequence mining programs MaxNaive, and MaxSequence are examined. First programs were tested using seven synthetic sequence databases described in table 6.2. The results are shown in figure 6.1. For each database two charts have been provided. The charts on the left compare the run times for MaxSequence (MS), MaxNaive (MN), and PrefixSpan (P). The charts on the right show the number of frequent sequences (Freq), number of Max sequences (Max), and the number of candidates generated by MaxSequence (Cand). The set of frequent sequences in fact is the candidate set for MaxNaive, so the number of frequent sequences also shows the number of candidates generated by MaxNaive. As it can be seen from the charts in all cases MaxSequence runs faster than MaxNaive. This speed becomes more noticeable as the minimum support decreases. Also as the minimum support decreases the difference between the number of frequent sequences, and the number of Max sequences increases. Number of the candidate sequences is always between the number of frequent sequences, and the number of Max sequences.

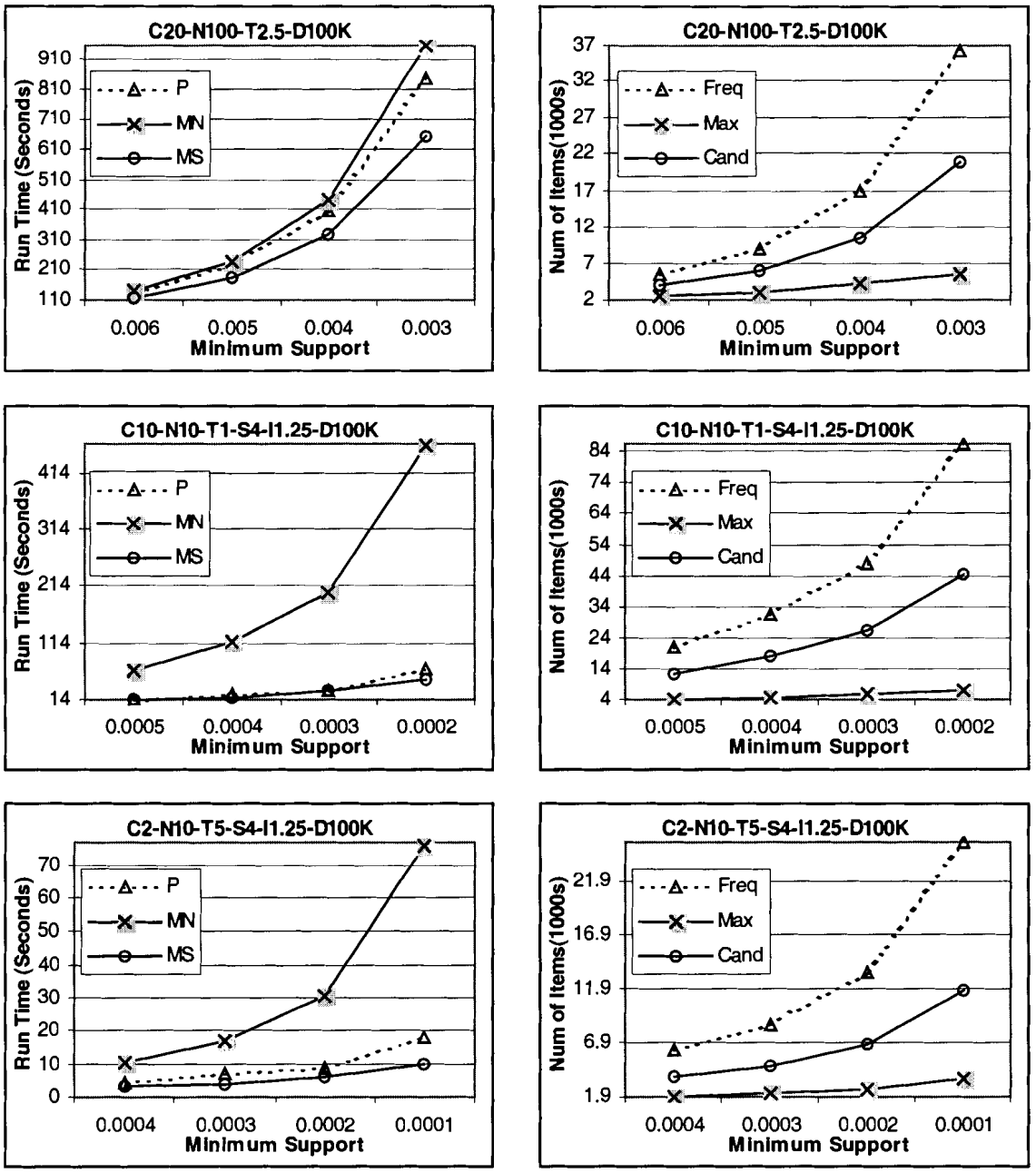

Figure 6.1 MaxSequence performance charts.

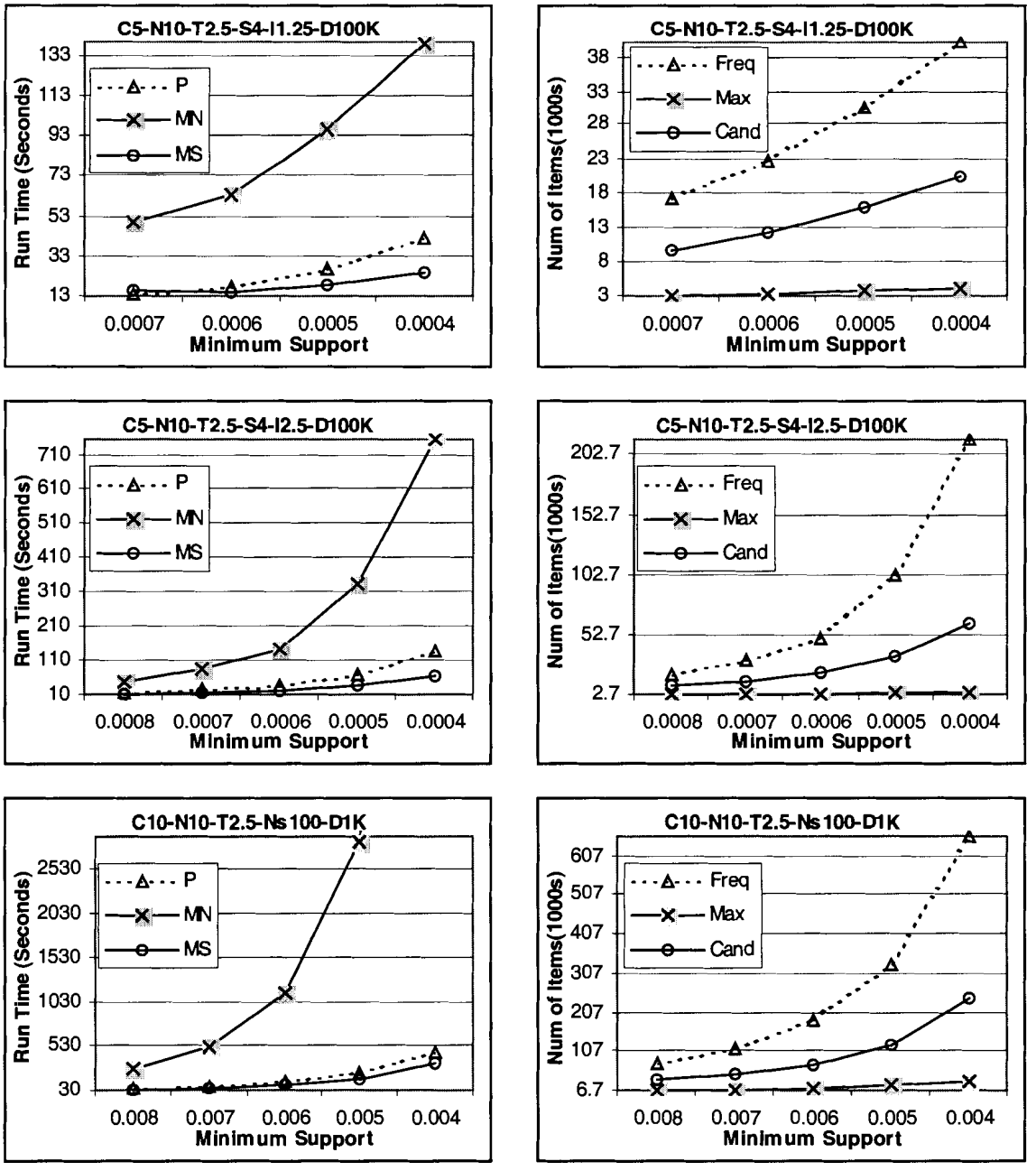

Figure 6.1 MaxSequence performance charts (continued).

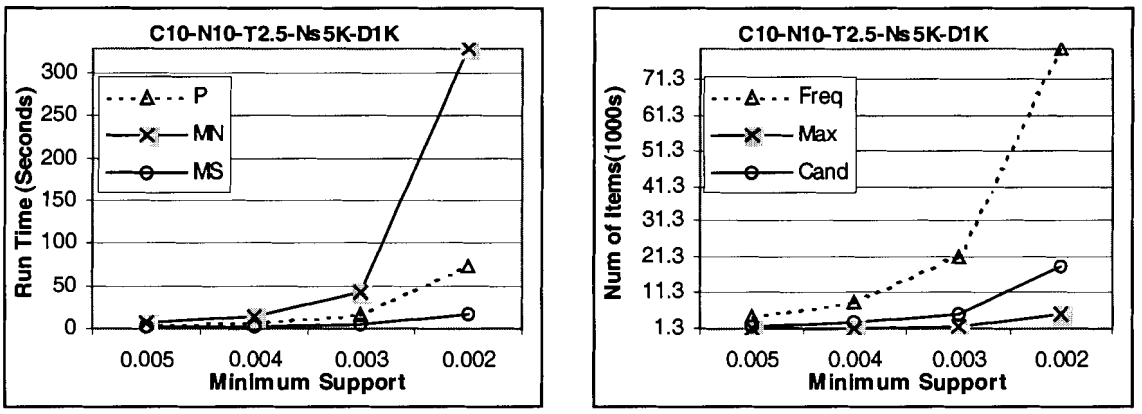

Figure 6.1 MaxSequence performance charts (continued).

In another experiment the performance of MaxSequence, with String Elimination, is compared with PrefixSpan using a real sequence dataset, ProductClicks. The results are shown in figure 6.2. As it can be seen from the charts the run time difference between two programs are significant in lower minimum supports. Also the number of frequent sequences in lower minimum support is in millions, comparing to the number of Max sequences in the same minimum support, which is in thousands.

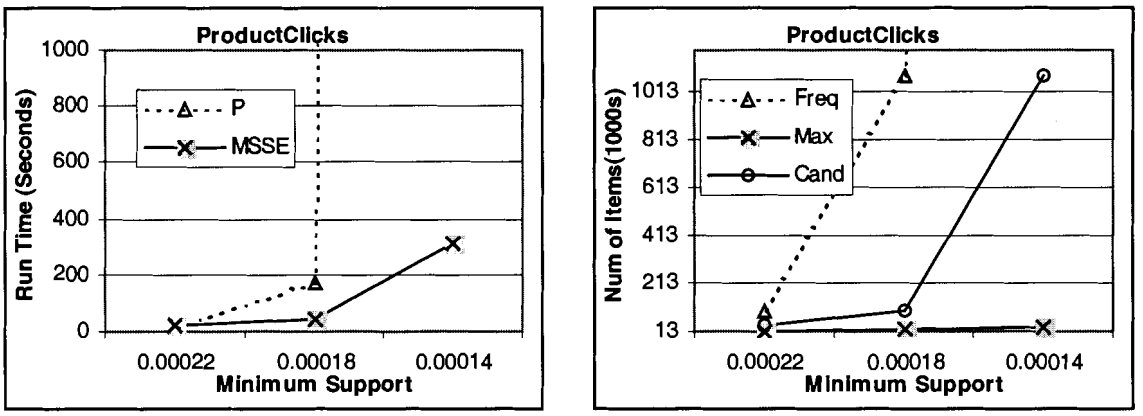

Figure 6.2 MaxSequence performance charts for ProductClicks.

6.2. Closed Sequence Mining Results

In this section the performance of Closed sequence mining programs ClosedNaive, and ClosedSequence are examined. Like previous section first the programs were tested using seven sequence databases described in table 6.2. The results are shown in figure 6.3. Again for each database two charts have been provided. The run time charts, and the number of candidate sequences charts. The charts on the left compare the run times for ClosedSequence (CS), ClosedNaive (CN), and PrefixSpan (P). The charts on the right show the number of frequent sequences (Freq), number of Closed sequences (Closed), and the number of candidates generated by ClosedSequence (Cand). The set of frequent sequences in fact is the candidate set for ClosedNaive, so the number of frequent sequences also shows the number of candidates generated by ClosedNaive. ClosedSequence performs much faster than the ClosedNaive, and the difference becomes more apparent as the minimum support decreases. Also as the minimum support decreases the difference between the number of frequent sequences, and the number of Closed sequences increases. Number of the candidate sequences is always between the number of frequent sequences, and the number of Closed sequences, but, as it can be guessed, the difference between the numbers of candidates generated by two programs is less than the case in Max sequence mining.

As it can be seen from comparing the run time charts in this section, with the ones from the previous section, in some cases the ClosedSequence program runs faster than the MaxSequence. This is because of the optimizations that were done during implementation for ClosedSequence, and is not related to the algorithms.

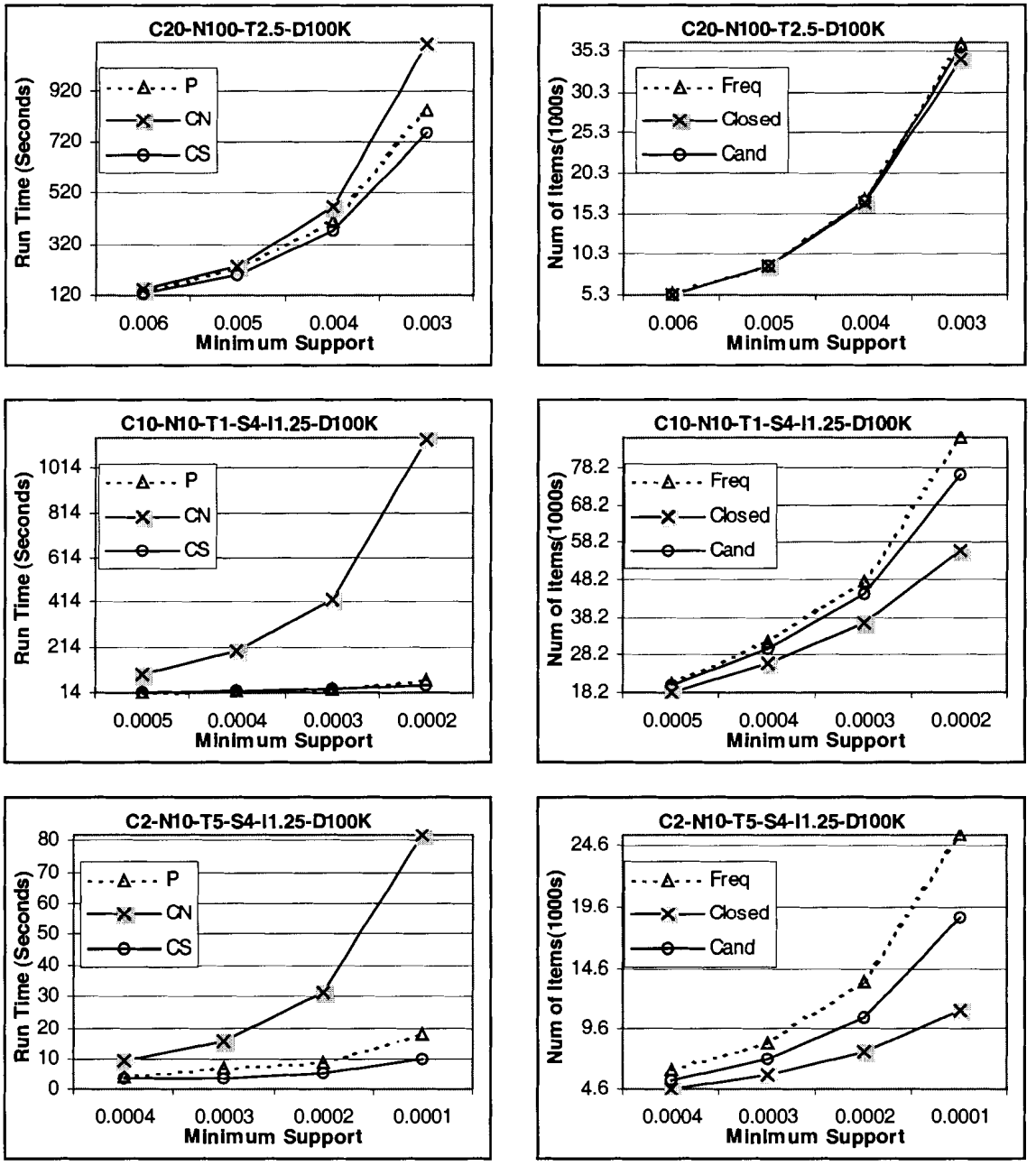

Figure 6.3 ClosedSequence performance charts.

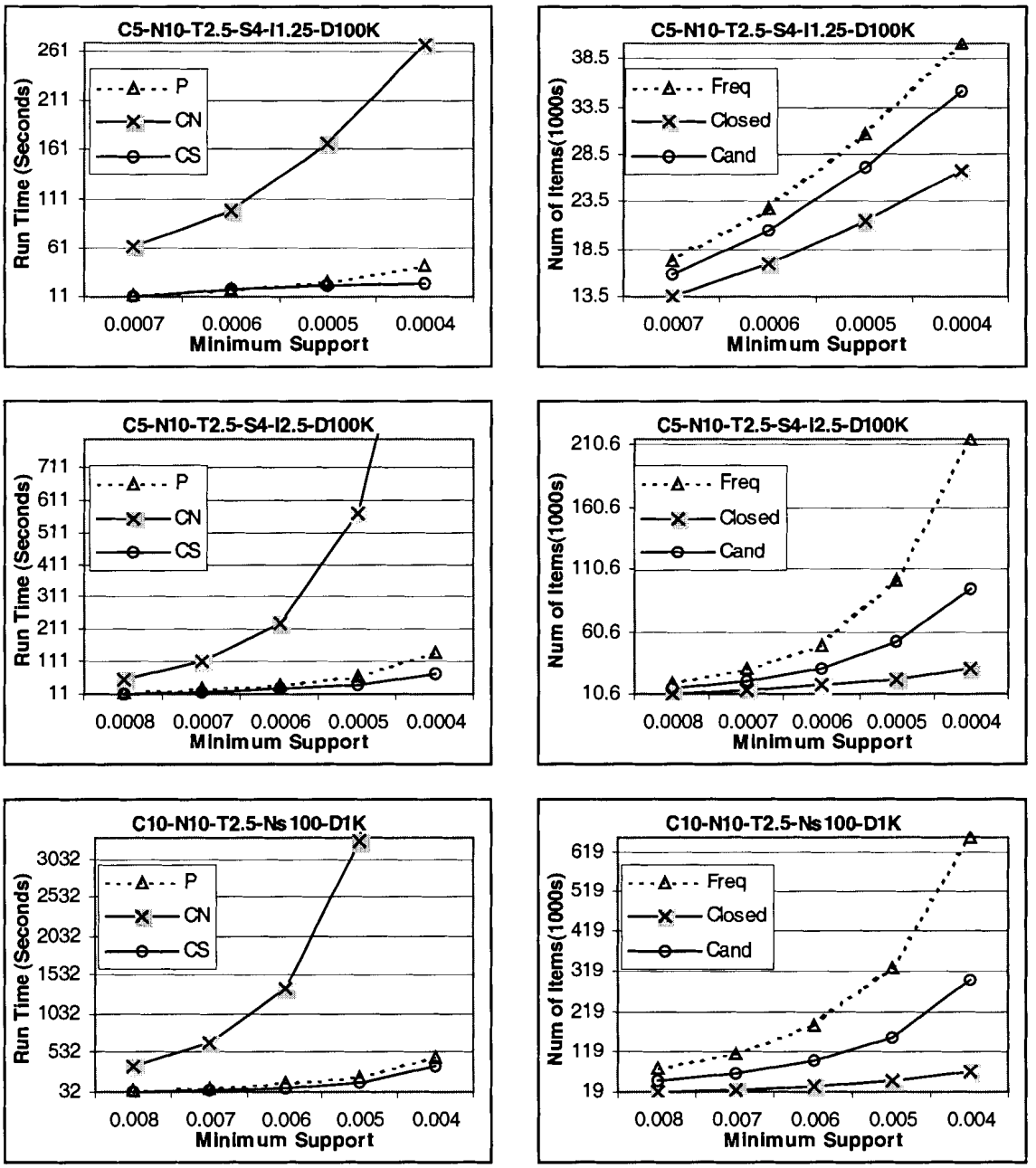

Figure 6.3 ClosedSequence performance charts (continued).

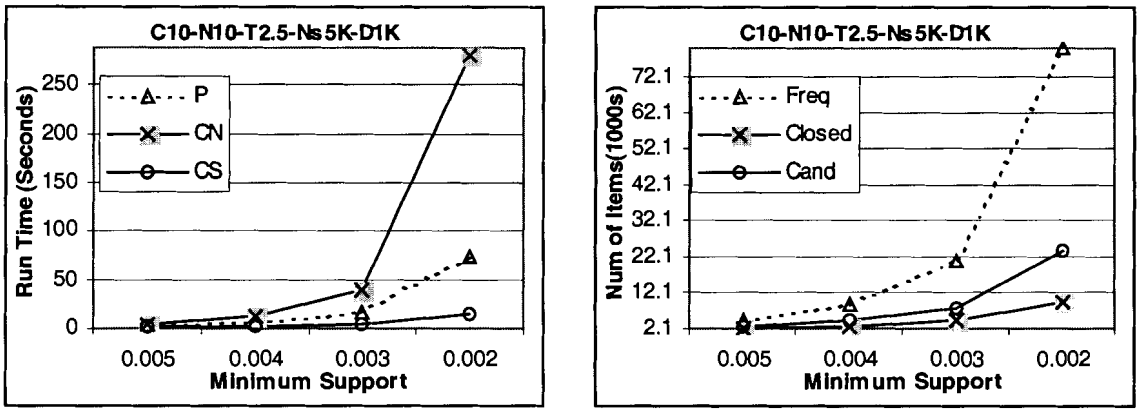

Figure 6.3 ClosedSequence performance charts (continued).

As another experiment the performance of ClosedSequence is compared with PrefixSpan using a real sequence dataset, ProductClicks. The results are shown in figure 6.2. As it can be seen from the charts, the run time difference between two programs becomes more noticeable as the minimum supports decreases. Also the number of frequent sequences in lower minimum support is in millions, comparing to the number of Closed sequences in the same minimum support, which is in thousands.

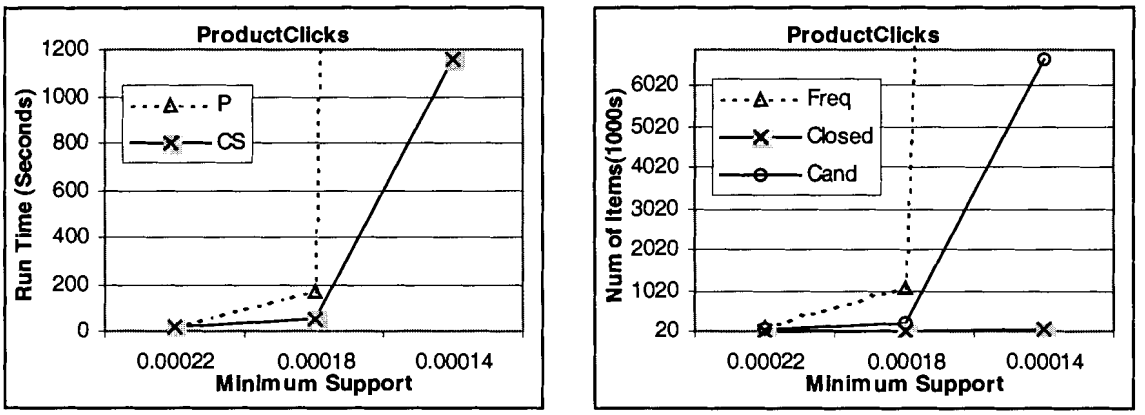

Figure 6.4 ClosedSequence performance charts for ProductClicks.

6.3. Effects of Optimization Methods

In this section we will study the effects of the optimization methods Common Prefix Detection, Max-Tree, Closed-Tree, and String Elimination. Different programs, described in table 6.3, are used to mine Max, and Closed sequences in the synthetic sequence database C5-N10-T2.5-S4-I2.5-D100K. Figure 6.5 shows the effect of String Elimination in Max sequence mining. In this chart we compare the run time for MaxSequence without String Elimination, MS, and MaxSequence with String Elimination, MSSE. As it can be seen from the chart, MSSE has a better performance than MS.

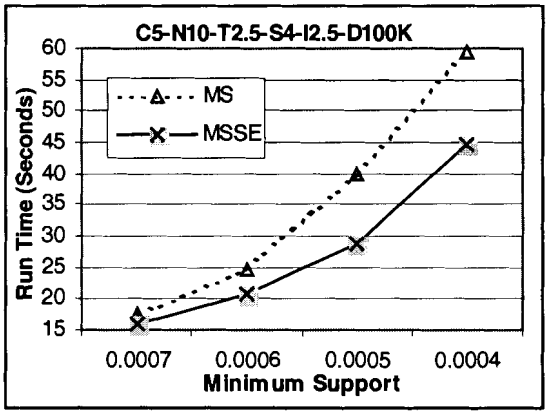

Figure 6.5 Effect of String Elimination.

Figure 6.6 shows the effects of using Max-Tree, and Closed-Tree in MaxSequence, and ClosedSequence programs. The chart on the left compares the run time of MaxSequence, which uses Max-Tree, MS, with the run time of a version of MaxSequence that uses a linear list instead of Max-Tree, MSNMT. The chart on the right shows the run time of ClosedSequence, which uses Closed-Tree, CS, and the run time of a version of ClosedSequence that uses a linear list instead of Closed-Tree, CSNCT. Both charts show the performance will increase significantly using Max-Tree, and Closed-Tree, instead of using linear lists.

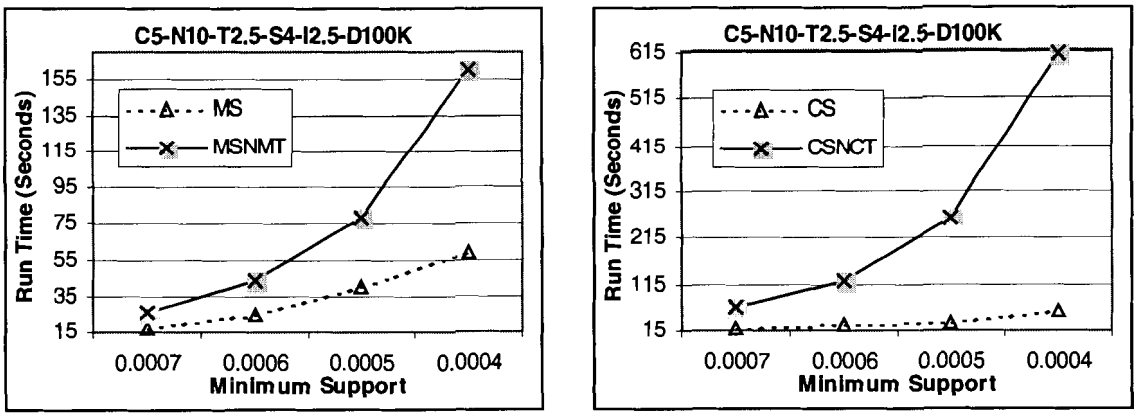

Figure 6.6 Effects of using Max-Tree, and Closed-Tree.

The effect of Common Prefix Detection is shown in the charts of the figure 6.7. The charts on the first row show this effect for Max sequence mining. On the left hand chart the run time of MaxSequence, MS, is compared with the run time of a version of MaxSequence, which does not use Common Prefix Detection, MSNP. This chart shows that MS outperforms MSNP significantly. The reason for this can be seen in the right hand chart that shows the number of candidates generated by MS is much less than the number of candidates generated by MSNP. The charts on the second row of the figure 6.7 show the effect of Common Prefix Detection in Closed sequence mining. On the left hand chart the run time of ClosedSequence, CS, is compared with the run time of a version of ClosedSequence called CSNP, which does not use Common Prefix Detection. This chart shows that CS outperforms CSNP significantly. The reason for this again can be seen in the right hand chart that shows the number of candidates generated by CS is much less than the number of candidates generated by CSNP.

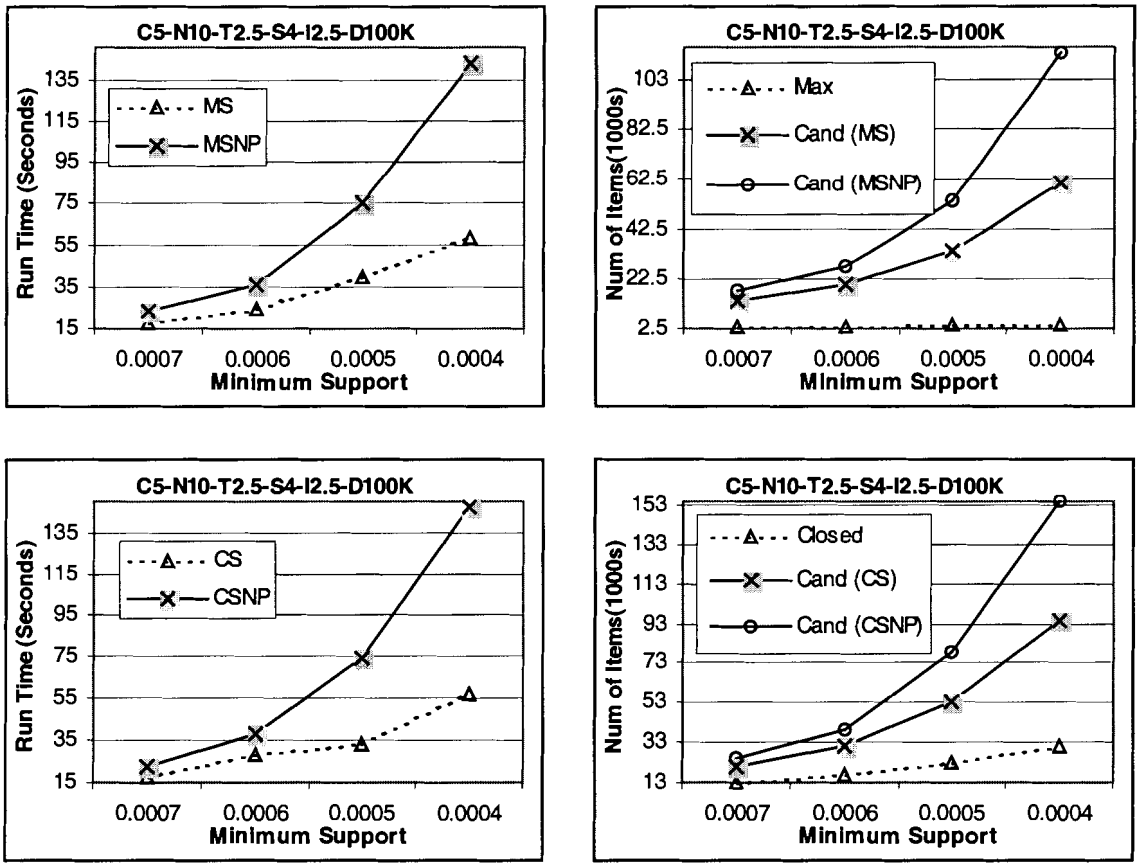

Figure 6.7 Effect of Common Prefix Detection.

Chapter 7. Conclusion and Future Work

Frequent Max, and Closed sequential pattern mining problems were formally defined in this thesis, and later different programs were suggested to solve these problems. The main idea for these programs is to generate some candidate sequences, and then keep only the Max, or Closed ones. Some ideas were suggested to decrease the number of candidate sequences generated, and also some data structures were developed to be able to keep more sequences in the memory, and to be able to compare the sequences more efficiently. According to the tests performed, the MaxSequence, and ClosedSequence programs seem to have an acceptable performance. The Prefix Detection feature in these programs might slow them down a little bit in general cases, but it can improve the performance drastically in extreme cases.

These results give an idea for future research. What if the program can decide to either use, or not use the Prefix Detection. This can be done based on different factors. One way would be to find some measure for database density, and based on this measure decide to use, or not use the Prefix Detection (a good measure for database density can be used in many other applications). Another approach could be to make a decision based on the minimum support value, and the number of the sequences in the projected database. As the number of sequences decreases in a projected database, the chance of having a common prefix among them increases, so the program can turn the Prefix Detection on, or off based on the number of sequences in the database. This threshold value could be related to the number of sequences in the original database, and the minimum support value.

Another research idea would be to scale up this process. The result set for these programs are kept in the main memory. Although these sequences are kept in the compressed form, using Max-Tree and Closed-Tree, but this can cause problem in some extreme cases, or when there is a restriction on the size of the usable memory. Max-Tree, and Closed-Tree

could be modified for these extreme cases, in a way that they can be stored on the disk, without losing their efficiency in checking new sequences.

Bibliography

[1] R. Agrawal, and R. Srikant: "Mining Sequential Patterns", *Proc. of the Int'l Conference on Data Engineering (ICDE)*, Taipei, Taiwan, March 1995.

[2] R. Srikant , and R. Agrawal: "Mining Sequential Patterns: Generalizations and performance improvements", In *Proc. 5th Int'l Conference Extending Database Technology (EDBT)*, Avignon, France, March 1996.

[3] W. Li: "Classification Based on Multiple Association Rules". M.Sc. Thesis, Simon Fraser University, April 2001.

[4] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth", *Proc. 2001 Int. Conf. on Data Engineering (ICDE'01),* Heidelberg, Germany, April 2001.

[5] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu, "FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining", *Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00),* Boston, MA, Aug. 2000.

[6] M. J. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences", in *Machine Learning Journal*, special issue on Unsupervised Learning (Doug Fisher, ed.), pages 31-60, Vol. 42 Nos. 1/2, Jan/Feb 2001.

[7] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases.", *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 207-216, Washington, D.C., May 1993.

[8] M. J. Zaki, and C. Hsiao, "CHARM: An Efficient Algorithm for Closed Association Rule Mining", in *Technical Report 99-10*, Computer Science, Rensselaer Polytechnic Institute, 1999.

[9] J. Pei, J. Han, and R. Mao, "CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets.", *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery 2000*, pages 21-30, Dallas, TX, 2000.

[10] *IBM Quest Data Mining Project Synthetic Data Generation Program*: http://www.almaden.ibm.com/cs/quest/syndata.html.

[11] K. Wang, and J. Tan, "Incremental Discovery of Sequential Patterns.", *In 1996 ACM SIGMOD Data Mining Workshop: Research Issues on Data Mining and Knowledge Discovery (SIGMOD'96)*, pages 95-102, Montreal, Canada, May 1996.

[12] R. Agrawal, and R. Srikant, "Fast algorithms for mining association rules in large databases.", In Research Report RJ 9839, IBM Almaden Research Center, San Jose, CA, June 1994.

[13] H. Mannila, H. Toivonen, and A. I. Verkamo, "Efficient algorithms for discovering association rules.", In *Proc. AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94)*, pages 181-192, Seattle, WA, July 1994.

[14] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "Parallel algorithm for discovery of association rules.", *Data Mining, and Knowledge Discovery*, 1:343-374, 1997.

[15] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation.", *Proc. 2000 ACM SIGMOD Int. Conf. Management of Data (SIGMOD'00)*, pages 1-12, Dallas, TX, May 2000.

[16] J. Pei, and J. Han, "Can We Push More Constraints into Frequent Pattern Mining?", *Proc. 2000 Int. Conf. on Knowledge Discovery and Data Mining (KDD'00)*, Boston, MA, August 2000.

[17] K. Wang, Y. He, and J. Han, "Mining Frequent Itemsets Using Support Constraints", *Proc. 2000 Int. Conf. on Very Large Data Bases (VLDB'00)*, Cairo, Egypt, September 2000.

[18] M. N. Garofalakis, R. Rastogi, and K. Shim, "Mining Sequential Patterns with Regular Expression Constraints", *IEEE Transactions on Knowledge, and Data Engineering*, volume 14, number 3, pages 530-552, May/June 2002.

[19] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. "KDD-Cup 2000 organizers' report: Peeling the onion." *SIGKDD Explorations*, volume 2, issue 2, pages 86-98, December 2000. http://www.ecn.purdue.edu/KDDCUP.