COBRA: Closed Sequential Pattern Mining Using Bi-phase Reduction Approach

Kuo-Yu Huang[1], Chia-Hui Chang[2], Jiun-Hung Tung[1] and Cheng-Tao Ho[1]
Department of Computer Science and Information Engineering,
National Central University, Chung-Li, Taiwan 320
[1]{want,ginhong,ctho}@db.csie.ncu.edu.tw, [2]chia@csie.ncu.edu.tw

Abstract

Sequential pattern mining aims to find frequent patterns (guarded by a minimum support) in a database of sequences. As the support decreases the number of sequential patterns will increase rapidly. Therefore, a new trend is to mine closed sequential patterns, i.e. patterns which have no super-patterns with the same support in the database. Since mining closed sequential pattern has the same capability as mining the complete set of sequential patterns while reduces redundant patterns to be generated and stored, it is much economical and beneficial. In this paper we propose a novel approach which extends a frequent sequence with closed itemsets instead of single items. The motivation is that closed sequential patterns are composed of only closed itemsets. Hence, unnecessary item extensions which generates non-closed sequential patterns can be avoided. Furthermore, we propose LayerPruning *which removes unnecessary enumeration by comparing candidate prefixes in the same layer rather than comparing one prefix against historical ones. Experimental evaluation shows that the proposed approach is two orders of magnitude faster than previous works with a modest memory cost.*

1 Introduction

Sequential pattern mining is a fundamental data mining task that has broad applications, including user behavior analysis, network intrusion detection and tandem repeats in DNA sequences. Ever since Argawal et al. [13, 14] introduced the concept of sequential pattern mining in 1995, this problem has received a great deal of attention [1, 2, 3, 4, 5, 12, 19]. Mining sequential pattern is more complex than frequent itemsets, since the permutations of items needs to be considered. Thus, instead of mining the complete set of frequent sequential patterns, we have stronger motive to mine closed sequential patterns, i.e. those containing no super-sequence with the same support. Mining closed sequential patterns

not only reduce the number of sequences presented to users but also increase the mining efficiency by pruning the enumeration space.

Although mining closed subsequences shares a similar problem setting with mining closed itemsets [6, 10], the techniques developed in closed itemset mining cannot work for frequent subsequence mining directly because subsequence testing requires ordered matching which is more difficult than simple subset testing. To the best of our knowledge, there are only two algorithms in closed sequential pattern mining, including CloSpan [18] and BIDE [16]. CloSpan takes the approach which generates a candidate set for closed sequential patterns and conducts post-pruning on it. The idea is that if a new discovered sequence s' is a sub-sequence or super-sequence of an existing sequence s and the projected database of s and s' is equal (closure checking), then we can stop searching any descendant of s' in the prefix search tree (thus pruning the search space) since for all γ the support of sequence $s' \diamond \gamma$ is equal to that of $s \diamond \gamma$. What makes the concept works is that the equivalence of the projected databases can be implemented by comparing the size of the databases. Furthermore, the size of the projected databases can be used as the hash key to improve subsequence/supersequence checking more efficiently. However, the candidate maintenance-and-test paradigm suffers the inherent drawback in scalability.

Therefore, Wang et al. propose an alternative solution without candidate maintenance. It adopts a sequence closure-checking scheme called BIDE. From definition, we know that if a sequence S = $< s_1, s_2, \ldots, s_n >$ is not a closed sequence, there must exist at lease an event e' which can be used to extend sequence S to a new sequence S' with the same support. The sequence S can be extended from the right most direction (after s_n), the left most direction (before s_1) or in the middle of the sequence (between s_i and s_{i+1}). If no such event exists, then S must be a closed sequence. Thus, the proposed BIDE scheme is to scan for common items from the sequence database, which might exist between s_i and s_{i+1}. As for search space pruning, they propose the *BackScan* pruning method to stop growing unnecessary patterns if the current prefix can not be closed. Again, they have defined the subsequences where the common items are searched for this BackScan closure checking. Although BIDE do not keep track of any historical closed sequential patterns (or candidate) for a new pattern's closure checking, it is a computational consuming approach since it needs multiple database scans for the bi-direction closure checking and the backscan pruning.

Both algorithms adopt the framework of PrefixSpan [12] which grows patterns by itemset extension

and sequence extension, i.e. the last transaction of the current sequence is extended with a frequent item in the same transaction (item extension) or different transaction (sequence extension). CloSpan prunes the search space by sub/super sequences with equivalent projected databases, while BIDE prunes the search space if the current pattern can be enumerated by other pattern. We refer to them as a *single phase reduction*. Different from previous works, in this paper we mine closed sequential pattern by a bi-phase reduction approach, *COBRA* (stands for **Clo**sed sequential pattern mining using **Bi**-phase **R**eduction **A**pproach). First, we mine closed frequent itemsets, then encode each of them as a unique C.F.I. $Code$ to replace the original database. Next, we generate closed sequential patterns only by sequence extension of C.F.I. $Code$s (closed itemsets). To make pruning more efficient, we propose $LayerPruning$ which removes unnecessary enumeration during the extensions of the same prefix pattern. Experimental evaluation shows that COBRA delivers order of magnitude performance improvements over the previous method BIDE.

The rest of this paper is organized as follows. We define the problem of closed sequential pattern mining in Section 2. Related works on sequential patterns and closed patterns are introduced in Section 3. Section 4 presents our algorithm. Experiments on both synthetic and real world data are reported in Section 6. Finally, conclusions are made in Section 7.

2 Problem Definition

Given a database SD of customer transactions, where each transaction consists of the following fields: customer-id, transaction-time, and the items purchased in the transaction. No customer has more than one transaction with the same transaction-time. Let $I = \{i_1, i_2, \ldots, i_N\}$ denote the set of items. A customer sequence can be represented by an ordered lists of itemsets, i.e., S=$<t_1, \ldots, t_n>$, where each itemset t_j is a non-empty subset of I, denoting the items bought in one transaction. The number of itemsets in a sequence is called the length of the sequence and a sequence with length l is called an l-sequence. A sequence α=$<a_1, \ldots, a_m>$ is a **sub-sequence** of another sequence β=$<b_1, \ldots, b_n>$, if and only if each a_j $(1 \leq j \leq m)$ can be mapped by b_{i_j} $(a_j \subseteq b_{i_j})$ and preserve its order $(1 \leq i_1 < i_2 < \ldots < i_m \leq n)$. We say β is super-sequence of α and β contains α. For example, sequence α=$<\{A, B\}, \{C\}, \{D, E\}>$ is a super-sequence of sequence β=$<\{A\}, \{D\}>$, since the pattern $\{A\}$ ($\{D\}$, resp.) is a subset of $\{A, B\}$ ($\{D, E\}$, resp.). On the contrary, γ=$<\{C, D\}>$ is not a sub-sequence of α, since the pattern $\{C, D\}$ can not be mapped to any itemset in α.

Table 1. An example sequence database SDB

SID	Sequence
1	$(C)(A,B,C)(B,C)(A,B,C)$
2	$(B,C)(A,B,C)(C)$
3	$(B,C)(A)(D,F)(A,B,C)(C)$
4	$(C)(A)(D,E)(A,B,C)$

A sequence database $SD = \{S_1, \ldots, S_{|SD|}\}$ is a set of sequences. Each sequence is associated with an sid. $|SD|$ represents the number of sequences in the database SD. The **absolute support** of a sequence α in a sequence database SD is the number of sequences in SD which contain α, and the **relative support** is the percentage of sequences in SD that contain α. Without loss of generality, we use the absolute support for describing the algorithm while using the relative support to present the experimental results.

Given two sequences α and β. If α is a super-sequence of β and their supports are the same, we say α **absorbs** β. A sequential pattern β is a **closed sequential pattern** if there exists no proper sequence α that absorb β. The problem of closed sequential pattern mining is formulated as follows: given a minimum support level $minsup$, our task is to mine all closed sequential patterns in the sequence database with support greater than $minsup$, i.e. the **frequent** sequential patterns.

Example 2.1 *Table 1 shows a sequence database SDB as our running example. Let $minsup$ be 3. The complete set of frequent closed sequences consists of only six sequences: {<(B,C) (A,B,C)>:3, <$(B,C)(B,C)(C)$>:3, <$(C)(A,B,C)(C)$>:3, <$(C)(A,B,C)$>:4, <$(C)(A)(A,B,C)$>:3, <$(C)(A)$ (C)>:4}, while the whole set of frequent sequences consists of 55 sequences (The value after colons denotes the support count). Most of the frequent sequences are absorbed by the closed sequences. For example, frequent sequence <$(B)(A,B,C)$>:3 is absorbed by <$(B,C)(A,B,C)$>:3.*

To make connection between closed itemsets with closed sequential patterns, we define transaction support and sequence support of an itemset as follows. The transaction support of an itemset ρ is defined as the number of transactions that contain ρ while the sequence support of ρ is the number of sequences that contain the 1-sequence ρ. As usual, an itemset ρ is closed if there exist no superset of ρ with the same transaction support. However, an itemset ρ is frequent in a sequence database SD if the sequence support of ρ is greater than $minsup$. Thus, ρ is an frequent closed itemset if the sequence support is

greater than $minsup$ and there exists no superset with the same transaction support.

Example 2.2 *Given $minsup = 3$, all subsets of $\{A, B, C\}$ are frequent in the example sequence database in Table 1 since each itemset has sequence support 4. However, only $\{A\}$, $\{C\}$, $\{B, C\}$, and $\{A, B, C\}$ are frequent closed itemsets. Itemset $\{B\}$ is not a closed itemset since it has the same transaction support 8 as itemset $\{B, C\}$. Similarly, itemsets $\{A, B\}$ and $\{A, C\}$ are absorbed by $\{A, B, C\}$ since they have the same transaction support 5.*

3 Related Works

The problem of mining sequential patterns was first introduced in [13] by Agrawal and Srikant, who also proposed the famous GSP (**G**eneralized **S**equential **P**attern) algorithm based on the Apriori property [14]. The GSP algorithm applies a breadth-first enumeration to generate candidate patterns and scans the horizontal database for verification. However, in situations with prolific frequent patterns, long patterns, or quite low $minsup$ thresholds, an Apriori-like algorithm may suffer from handling a huge number of candidate sets and multiple database scans. To overcome these drawbacks, Han et al. propose PrefixSpan [12] that adopts a depth-first enumeration and scans the projected database (also in horizontal data format) to extend longer patterns. The general idea of this pattern growth method is to recursively project the database into a set of smaller databases with respect to a frequent pattern and extend the pattern by exploring only frequent items in the projected partition. Therefore, they can avoid expensive candidate generation and repeated database scans for support counting.

In addition to algorithms based on horizontal formats, Zaki proposed a vertical-based algorithm called SPADE [19]. SPADE utilizes combinatorial properties to decompose the original problem into smaller sub-problems that can be independently solved in main memory using efficient lattice search techniques and simple *join operations*. SPAM [1] is similar to SPADE except that it employs a vertical bitmap representation. It is more efficient than PrefixSpan and SPADE, but it consumes more space for vertical bitmap maintenance.

Since pattern mining may generate a huge of patterns, it reduces not only the effectiveness but also the efficiency of mining. Therefore, Pasquier et al. [9] have proposed to mine closed patterns for frequent itemsets. Several efficient algorithm are proposed for closed frequent itemset recently, including A-Close [9], CLOSET [11], CHARM [20] and CLOSET+ [17]. Although, some closure checking and pruning

strategies in closed frequent itemset can be extended for closed sequential pattern, it only prunes non-closed patterns at the item extension step. So far, there are only two algorithms proposed for closed sequential pattern mining, CloSpan and BIDE.

CloSpan applies a novel concept called the equivalence of projected databases for backward sub-pattern/superpattern checking to prune the search space. Upon the generation of a frequent sequence, CloSpan has to maintain the frequent sequences generated so far unless they are verified to be non-closed by subpattern/superpattern checking. Different from CloSpan, which is based on the relationships among the newly found frequent pattern and some already mined closed patterns, BIDE proposes a bi-directional scan to remove non-closed pattern without candidate maintenance and devises a $BackScan$ pruning method to stop unnecessary enumeration of prefix sequence quickly. Since the criteria for $BackScan$ pruning is a common item for a particular period (called semi-maximum periods) in all sequences, the checking can terminate as early as the intersection becomes empty. Although BIDE reduces the memory requirement for candidate maintenance, it requires more computation and multiple data scans in backward checking. In addition, the simple set intersection of $ScanSkip$ is useless if the approach is applied to complex sequences (composed by itemsets, e.g. Table 1).

Note that both CloSpan and BIDE followed the same enumeration strategy: patterns are generated based on the sequence lexicographic order by performing item extension (or I-step, denoted by $P \diamond_i item = <s_1, \ldots, s_n \cup item>$) and then sequence extension (or S-step, denoted by $P \diamond_s item = <s_1, \ldots, s_n, item>$). However this pattern-growth strategy has two drawbacks: duplicate item extensions and expensive matching cost, as described below.

1. **Duplicate item extensions**: Consider a database of two sequences S_1=$<\{A_1, B_2\}, \{A_3, B_4, C_5\}, \{A_6, B_7\}>$ and S_2=$<\{A_8, B_9, C_{10}\}, \{A_{11}, B_{12}\}, \{A_{13}, B_{14}\}>$. To find the closed sequence $<\{A, B\}, \{A, B\}, \{A, B\}>$, we need three item extensions (e.g. $\{A\} \diamond_i \{B\}$), which are duplicate and unnecessary since $\{A\}$ itself is not closed (wherever A occurs B also occurs). If we can do sequence extension by adding $\{A, B\}$ instead of single item $\{A\}$, then such duplicate item enumeration can be avoided.

2. **Expensive Matching Cost**: The matching cost refers to the process of finding the locally frequent items in the projected database, especially for item extension. For example, the projected position for prefix $\{A, B\}$ are B_2 and B_9, i.e. the last transaction where the first instance of the prefix

Code	C.F.I.	FML	LocationList (SID,TID)
#1	ABC	2, 6, 10,14	(1,2), (1,4), (2,6), (3,10), (4,14)
#2	BC	2, 5, 8, 14	(1,2),(1,3), (1,4), (2,5), (2,6), (3,8), (3,10), (4,14)
#3	A	2, 6, 9, 13	(1,2), (1,4), (2,6), (3,9), (3,10), (4,13), (4,14)
#4	C	1, 5, 8, 12	(1,1), (1,2), (1,3), (1,4), (2,5), (2,6), (2,7), (3,8), (3,10), (3,11), (4,12), (4,14)

Figure 1. Vertical-based LocationList and FML

occurs. Extendable items for I-steps are then searched after these positions by matching with the prefix. Thus, we can find locally frequent items C which occurs not only in S_2 at C_{10} but also in S_1 at C_5 where the prefix $\{A, B\}$ occurs in the same transaction.

In this paper, we have come up with a novel approach which conducts only sequence extensions by adding frequent closed itemsets to overcome these drawbacks. Frequent closed itemsets, as proved in the next section, are in fact the basic components of frequent closed sequences. They can be used to remove duplicate item enumeration as well as to reduce the matching cost for finding locally frequent items for I-extension.

4 Algorithm Overview

In this section, we present an important observation and prove that a frequent closed sequential pattern is composed of only frequent closed itemsets. Thus, we devise a bi-phase reduction approach which mines frequent closed itemsets first and enumerate frequent closed sequential patterns by conducting sequence extensions. Before introducing the pruning strategy, we first define some terms.

Definition 4.1 *Given a sequence $S = <s_1, \ldots, s_n>$, the **First Matched Transaction (FMT)** of a 1-sequence $<p_1>$ is defined as the transactional ID of the first instance of the itemset p_1. Recursively, we can define the FMT of a $(m+1)$-sequence $<p_1 \ldots p_m p_{m+1}>$ from the FMT of the m-sequence $<p_1 \ldots p_m>$ as (the transaction ID of) the first appearance of itemset p_{m+1} which also occurs after the FMT of the m-sequence $<p_1 \ldots p_m>$. Given a sequence database SD (each transaction in SD has a unique ID), the **First Matched transaction List (FML)** of a prefix sequence α=$<p_1 \ldots p_n>$ is defined as the list of first matched transactions of the sequences in SD w.r.t. α. Similarly, the **SID List** of α is a list of sequence IDs that support α.*

For example, in the database SD of three sequences $S_1 = <\{A, C\}_1, \{A, B, C\}_3, \{A, B\}_4>$, $S_2 = <\{A\}_2, \{B\}_8>$ and $S_3 = <\{A, B\}_5, \{A, C\}_6, \{B, C\}_{10}>$, where the subscripts denote the transaction

TID	1	2	3	4	5	6	7	8	9	10	11	12	13	14
SID	1	1	1	1	2	2	2	3	3	3	3	4	4	4
Code	#4	#1	#2	#1	#2	#1	#4	#2	#3	#1	#4	#4	#3	#1
		#2	#4	#2	#4	#2		#4		#2				#2
		#3		#3		#3				#3				#3
		#4		#4		#4				#4				#4

Figure 2. Horizontal Encoded Database EDB

IDs, the FMLs of the prefix sequence $<\{A, B\}>$ and $<\{A\}\{B\}>$ are $\{3, 5\}$ and $\{3, 8, 10\}$, respectively, while their SID Lists are $\{1, 3\}$ and $\{1, 2, 3\}$, respectively. Note that we can look up the corresponding sequence IDs given the transaction IDs in the FML.

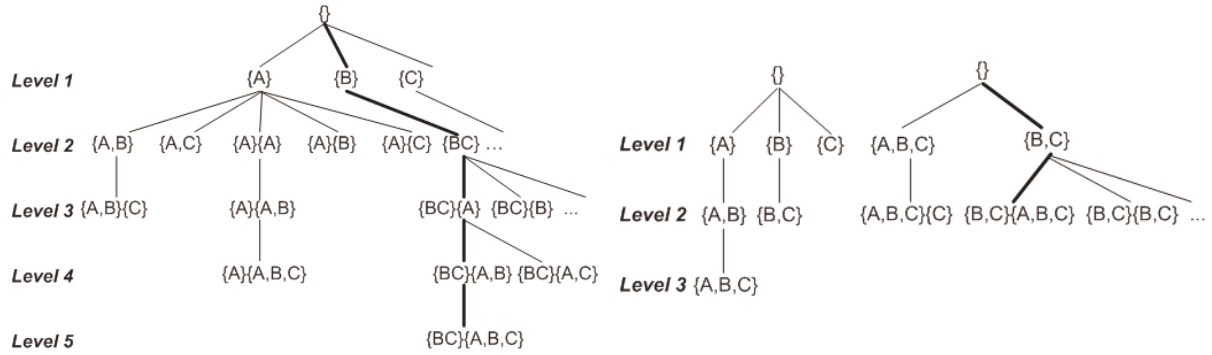
Given an itemset p, let $c(p)$ denote the closed itemset which contains p and has the same transaction support as p. If p is closed, then $c(p) = p$. By definition, $c(p)$ and p have the same transaction support and the FML are the same (denoted as $p.FML = c(p).FML$).

Lemma 4.1 *Given three sequential patterns α, β and γ, if $\alpha.FML$ = $\beta.FML$ then $\alpha \diamond_s \gamma.FML = \beta \diamond_s \gamma.FML$ and $\alpha \diamond_s \gamma.SIDList = \alpha \diamond_s \gamma.SIDList$ (Definition 4.1).*

Theorem 4.1 *A closed sequential pattern is composed of only closed itemsets.*

Proof 4.1 *Assume $\alpha = p_1 \diamond_s \ldots \diamond_s p_n$ is a closed sequential pattern, but some of the p_is are non-closed itemsets. Consider a sequential pattern $\beta = c(p_1) \diamond_s p_2 \diamond_s \ldots \diamond_s p_n$, $\alpha.SIDList = \beta.SIDList$ since $p_1.FML = c(p_1).FML$ (Lemma 4.1). Recursively, we can find a sequential pattern $\delta = c(p_1) \diamond_s \ldots \diamond_s c(p_n)$ such that $\alpha.FML = \delta.FML$. Therefore, α is not a closed sequential pattern. We thus have a contradiction to the original assumption that α is a closed sequential pattern and thus conclude that "all closed sequential patterns α are composed of only closed itemsets."*

Theorem 4.1 is an important property as it provides a different view of mining closed sequential patterns. Instead of extending a prefix by I-steps and S-steps alternatively, we can mine closed frequent itemsets before mining closed sequential patterns and extends a prefix sequence by only S-steps. Therefore, we have come up with a three phase algorithm. In the first phase, we find all frequent closed itemsets and denote each of them by a unique C.F.I. $code$. To avoid the need to match closed frequent itemsets in a sequence in the enumeration phase, the original database is transformed into another database where



(a) Search space of original enumeration (b) Search space of our enumeration

Figure 3. Enumerative Trees

the items in each sequence are replaced by C.F.I. $code$s that are contained in the transactions. Finally, the closed sequential patterns are enumerated in the third phase.

To illustrate, the example database SDB (Table 1) can be transformed into Figure 2 given the C.F.I. $code$s shown in Figure 1. This transformation retains the horizontal format of the original database. Note that the transactions are renumbered to eliminate empty transactions due to the removal of non-frequent items (e.g. D, E, F). Figure 1 also shows the location lists of each closed frequent itemset, which represent the vertical format of the original database.

We refer this as a bi-phase reduction approach since we mine C.F.I. for first phase reduction then mine closed sequences for second phase reduction. This approach not only reduces the search spaces and duplicate combinations but also avoids the matching costs in item extension process. As shown in Figure 3, the search space for our enumeration tree is much smaller than that of the typical lexicographic tree. Obviously, the longest path (the thick lines) in our enumerative strategy is shorter than that of the typical strategy.

A similar framework has also been adopted in [15] for inter-transaction association mining. However, applying such a framework in closed pattern mining is much more economic than regular pattern mining since the number of frequent itemsets are larger than that of frequent closed itemsets. In the next section, we will discuss how to further prune the search space by **LayerPruning** and **ExtPruning**.

4.1 Pruning Strategies

Although the number of closed itemsets can be larger than the number of items, which seems to harm the mining process, lots of them can be ignored without consideration by layer pruning. As a contrast to previous works which only prune a branch of a non-closed pattern, layer pruning removes several non-closed branches at once and reduces the costs in pattern checking. Before introducing the pruning strategy, we first define the order of two first match lists.

Definition 4.2 ***(The Order of FML)*** *Given two FMLs* $S_1.FML = \{a_1, a_2, \ldots, a_m\}$ *and* $S_2.FML = \{b_1, b_2, \ldots, b_n\}$ *($m \geq n$), we say that* $S_1.FML <_L S_2.FML$ *if and only if there exists* $i_1, i_2, ..., i_n$ *such that* $a_{i_j}.SID = b_j.SID$ *and* $a_{i_j} < b_j$ *for all* j *($1 \leq j \leq n$). The equal signs hold ($S.FML =_L S'.FML$) when* $m = n$ *and* $a_j = b_j$ *for all* j*, ($1 \leq j \leq m$).*

Example 4.1 *Consider the example database* SDB *again, Figure 1 shows the first matched transaction list (FML) for the frequent closed itemsets which are also 1-sequences. The FML for C.F.I.* $codes$ *#1, #2, #3, #4 are* $\{2,6,10,14\}$*,* $\{2,5,8,14\}$*,* $\{2,6,9,13\}$ *and* $\{1,5,8,12\}$*, respectively. The orders between these FMLs are* $\#1.FML >_L \#4.FML$ *and* $\#3.FML >_L \#4.FML$*.*

LayerPruning: For two C.F.I. p_1 and p_2 that can be a sequence extension of a prefix sequence α=$<s_1, \ldots, s_n>$ in form of $S_1 = \alpha \diamond_s p_1$ and $S_2 = \alpha \diamond_s p_2$, the LayerPruning works as follows:

1. If $S_1.FML <_L S_2.FML$, then remove p_2. Vice versa.
2. If $S_1.FML =_L S_2.FML$, then if (a) $p_1 \subseteq p_2$, then remove p_1; (b) $p_2 \subseteq p_1$, then remove p_2; (c) neither $p_1 \subset p_2$ nor $p_1 \supset p_2$, then remove both p_1 and p_2.

For instance in our running example, we can completely skip prefix $\#1$ and $\#3$ from root since $\#1.FML >_L \#4.FML$ and $\#3.FML >_L \#4.FML$. Thus, the LayerPruning technique removes non-closed patterns in the same layer since the pruning is invoked within a local search of a prefix pattern. The correctness of the pruning technique can be proven by the following lemma and theorems.

Theorem 4.2 *Let two C.F.I.* p_1 *and* p_2 *that can be a sequence extension of a prefix sequence* α=$<s_1, \ldots, s_n>$ *in form of* $S_1 = \alpha \diamond_s p_1$ *and* $S_2 = \alpha \diamond_s p_2$*. If* $S_1.FML <_L S_2.FML$*, then all extensions of* S_2 *must not be closed.*

Proof 4.2 *By definition (Definition 4.1), the FML of α is smaller than that of its extensions, therefore, $\alpha.FML <_L S_1.FML$. Since $S_1.FML <_L S_2.FML$, wherever $p2$ occurs, $p1$ will also occur in the interval between $\alpha.FML$ and $S_2.FML$. Thus, the super-sequence $S' = \alpha \diamond_s p_1 \diamond_s p_2$ of S_2 has the same FML as S_2, and $S'.SIDList = S_2.SIDLis$ (Lemma 4.1). Therefore, S_2 is not a closed sequential pattern.*

Theorem 4.3 *Let two C.F.I. p_1 and p_2 that can be a sequence extension of a prefix sequence α=$<s_1, \ldots, s_n>$ in form of $S_1 = \alpha \diamond_s p_1$ and $S_2 = \alpha \diamond_s p_2$, and $S_1.FML =_L S_2.FML$. (a) If $p_1 \subset p_2$, then all extensions of S_1 must not be closed. (b) If neither $p_1 \subset p_2$ nor $p_1 \supset p_2$, then all extensions of p_1 and p_2 must not be closed.*

Proof 4.3 *(a) First, S_1 is a subsequence of S_2 since p_1 is a subset of p_2. Second, S_1 and S_2 have the same support since $S_1.FML =_L S_2.FML$. Therefore, S_1 is not a closed sequential pattern.*

(b) Consider the sequential pattern $\beta = \alpha \diamond_s p_1 \diamond_i p_2$=$<s_1, \ldots, s_n, p_1 \cup p_2>$. Since $S_1.FML =_L S_2.FML$ and $\beta.FML =_L S_1.FML \cap S_2.FML$, we have $\beta.FML =_L S_1.FML =_L S_2.FML$. Therefore, for any extension S_1 and S_2 of α, there exists β, such that β is a super sequence of S_1 and S_2, and $\beta.SIDList = S_1.SIDList = S_2.SIDList$. Therefore, S_1 and S_2 are not the closed sequential pattern.

LayerPruning have successfully prune non-closed sequences during sequence extension step of a prefix sequence. However, there are still some non-closed sequential patterns that can be generated in different layer. Therefore, we need a checking step to remove non-closed sequential patterns, we refer to this pruning as ExtPruning.

ExtPruning: For two sequential pattern α and β, the rule of ExtPruning states that

1. If $\alpha.FML =_L \beta.FML$ and α is a super sequence of β, then remove β and vice versa.
2. If $Sup(\alpha) = Sup(\beta)$ and α is a super sequence of β, then β is not closed pattern, vice versa.

The first rule of ExtPruning holds according to Theorem 4.3, while the second rule follows the definition of closed sequential patterns.

4.2 COBRA: Design and Implementation

In this section, we discuss the implementation of the COBRA algorithm. COBRA can be outlined as three major phases: (I) Mining Closed Frequent Itemset; (II) Database Encoding; and (III) Mining

Procedure **COBRA(sequence database** SD**,** $minsup$**)**
1. **Call mCHARM() to find the set of all C.F.I.;**
2. **Associate each C.F.I. with an** $code$**,**
and let CS **denotes the set of** $code$**s.**
3. **Construct the encoded DB** EDB **using** CS**;**
4. CS **= LayerPruning(**CS**);**
5. **for each** $code_i$ **in** CS **do**
6. **cobraDFS(**$code_i, code.FML$**);**

Subprocedure **cobraDFS(**α**,** FML**)**
7. **Compute Extended List** EL **of** FML**;**
8. **if (** $|EL| < minsup$ **) then**
9. **ExtPruning(**α**); return;**
10. **end**
11. **if (** $|EL| < |FML|$ **) then**
12. **if (ExtPruning(**α**)) then return;**
13. LC **= Local Frequent Codes in** $\alpha.PDB$**;**
14. LC **= LayerPruning(**LC**);**
15. **if (** $|EL| = |FML|$ **) then**
16. FEI **= All** LC_i**s with** $|LC_i.FML| = |FML|$**;**
17. **if (** FEI **==** ϕ **) then**
18. **if (ExtPruning(**α**)) then return;**
19. **end**
20. **for each** LC_i **in** LC **do**
21. **cobraDFS(**$\alpha \diamond_s LC_i$**,** $LC_i.FML$**);**

Figure 4. COBRA Algorithm

Closed Sequential Pattern. Figure 4 shows the pseudo code of the COBRA algorithm. Line 1 calls a modified CHARM [20] to mine frequent closed itemsets. Line 2-3 associates each C.F.I. with a unique $code$ and constructs the encoded database EDB using the $code$s of the C.F.I. Line 4-21 mines the set of all frequent closed sequential patterns. Details are described below.

There are already many closed frequent itemset mining algorithms. We prefer using a vertical-based mining algorithm in the first phase (e.g., CHARM[20]) since the vertical format records the locations (TIDList) of C.F.I.s which can be used to construct the transformed database in the second phase. We have modified CHARM as memory-based $mCHARM$, which validates local frequent items to reduce

unnecessary combinations of existing frequent itemsets with nonlocal frequent items. This is done by maintain the horizontal-based database in memory and use the TIDLists (vertical-based) of frequent itemsets as index to accelerate the validation of local frequent items in the projected positions of the itemsets. Recall that frequent closed itemsets in a sequence database are defined by both sequence supports and transaction support, therefore, transaction ids are replaced by a 2-tuple (SID, TID) location to facilitate the counting of sequence supports and transaction supports.

In the second phase, we associate each C.F.I. with a unique $code$ and construct the encoded database in horizontal format based on the location lists of the C.F.I. Note that C.F.I.s are sorted by their length in a decreasing order such that super-sequences are generated earlier to reduce update cost in the third phase. Once the encoded database is constructed, we can release the memory space of $LocationList$ for all C.F.I.s. Furthermore, we can remove transactions without any frequent items to reduce the size of storage. Then, the first match transaction list for each C.F.I. (also the frequent 1-sequences) is constructed for the use in the third phase.

The mining process follows the idea of PrefixSpan to look for locally frequent (extendable) $code$s in the projected database of a prefix sequence. Starting with an empty sequence, the extendable $code$s are the frequent C.F.I.s. However, before the enumeration, we first apply the $LayerPruning$ strategy to remove unnecessary enumeration in the same layer (line 4). To reduce the cost of comparing any two FMLs (a total of $O(|C.F.I.|^2)$ comparisons), we devise a hash structure which uses Equation (1) as its hash function[1]. Equation (1) has more uniformly distributed keys than simple $|SIDList|$ can do. Only C.F.I.s that are hashed to the same bucket are compared to each other. Extendable C.F.I. that are not able to produce closed sequential patterns are then removed based on Theorem 4.2 and 4.3. In the pseudo code, the procedure $LayerPruning$, which implements the above idea, takes {#1, #2, #3, #4} as an input and returns $\{\#2, \#4\}$ since $\#4.FML <_L \#1.FML$ and $\#4.FML <_L \#3.FML$.

$$(|SIDList| + \sum_{Sid \in SIDList} Sid * pNo)\%HSize \tag{1}$$

In the procedure cobraDFS, with a new pattern α and its FML $\alpha.FML = \{t_1, \ldots, t_n\}$, we first compute the extended position list (EL) by looking at the next transaction of t_i, which has the same sequence id with t_i. For example, the EL of $code$ $\#2$ in Figure 1 is $\{3, 6, 9\}$ (transaction 15 is discarded

[1] pNo is chosen to be a prime number. $HSize$ is the size of the hash table.

for it does not have a sequence id as transaction 14). The number of transactions in the EL represents the largest support an extended sequence of α can have. Thus, if $|EL|$ is less than $minsup$, then we can skip all extensions of the prefix α (line 8-10); otherwise we do the extension of α (lines 11-21). In the later case, we compute the projected database of α (line 13) and find all locally frequent $code$s (denoted by LC). Again, before extension, $LayerPruning$ is applied to removes unnecessary $code$s (line 14). Formally, we define the extended list (EL) and projected database (PDB) of a pattern as follows.

Definition 4.3 *Given a sequence α and its $FML = \{t_1, \ldots, t_n\}$, the **Extended List (EL)** of α is defined as a list of extended position t'_i where $t'_i = t_1 + 1$ and $t'_i.SID = t_i.SID$.*

Definition 4.4 *Given the extended list of a sequential pattern α, with extended list $\alpha.EL$ = $\{t_1, \ldots, t_n\}$, the **Projected***

***Database (PDB)** of α is defined as $\alpha.PDB$ = $\{t'_1, \ldots, t'_n\}$ where $t'_i.SID = t_{i_j}.SID$ for some t_{i_j} and $t_{i_j} < t'_i \leq t_{|SD|}$, where $|SD|$ denotes the number of transactions in the extended databases.*

Definition 4.5 *Given a sequence α=$<s_1, \ldots, s_n>$, the **Forward Extended Itemset (FEI)** of α is defined as the set of extended $code$s of α which have the same SIDList as α, i.e. $\alpha.SIDList = \alpha \diamond_s p'_i.SIDList$.*

We output the new prefix sequence α only when it has the chance to be a closed sequential pattern. This includes the following three cases: (1) $|EL| < minsup$ (line 8) (2) $|EL| < |FML|$ (line 11-12) (3) $|FEL| = \phi$ (line 17-18). In the first case, no super-sequence of α can be generated as frequent patterns. In the second case, the supports of all super-sequences of α are less than α. In the third case, there are no extendable $code$s with the same support as α. This is equivalent to check for common $code$s that can be extended from the right direction (one of the two directions in BIDE). However, non closed sequential patterns still can be generated. Therefore, we should make a closure checking to verify if α is a closed sequential pattern or not. This is implemented by $ExtPruning$ which maintains the set of generated sequences.

Similar to $LayerPruning$, $ExtPruning$ also uses Equation 1 as the hash function. The hash table for $ExtPruning$ is called $CSTab$. A sequence α is only compared to sequences with the same SIDLists. To illustrate, assume $HSize$=3 and $pNo = 13$, we insert $\#2$ into closed since $|\#2.FML| > |\#2.EL|$. Thus, we insert $\#2$=$<\{A, B\}>$ into bucket 2 since the hash key $(4 + \sum(\#2.SIDList) * 13)\%3 = 2$. When sequence $\#4 \diamond_s \#1$ is generated, it is also hashed into bucket 2 for $\#4 \diamond_s \#1.FML =$

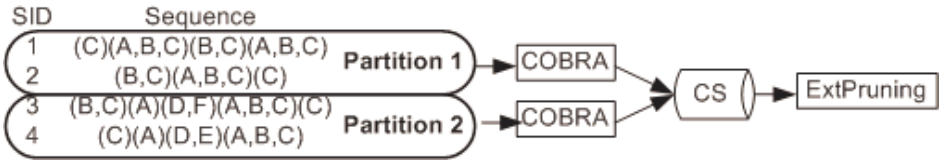


Figure 5. Horizontal-Based Partition

$\{2, 6, 10, 14\}$. Since $\#4 \diamond_s \#1 = <\{C\}\{A, B, C\}>$ is a super sequence of $<\{A, B\}>$ with the same support, $<\{A, B\}>$ is replaced by $<\{C\}\{A, B, C\}>$ and $ExtPruning$ return a value $False$. The return value of $ExtPruning$ indicates whether the extension of prefix α should go on. If α is a subsequence of an existing pattern β in the hash table and $\alpha.FML = \beta.FML$, then we simply discard α and return $True$ to stop the extension of prefix α (line 12,18).

Theorem 4.4 *The COBRA algorithm generates all closed sequential patterns.*

Proof 4.4 *First of all, the anti-monotone property "if a pattern is not frequent, all its super-patterns must be infrequent" is sustained for closed sequential patterns. According to Theorem 4.1, the search space composed of only closed frequent itemset covers all closed sequential patterns. COBRA's search is based on a complete set enumeration space. The only branches that are pruned as those that do not have sufficient support. The $LayerPruing$ only removes unnecessary enumerations (Theorem 4.2 and 4.3). On the other hand, $ExtPruning$ remove only non-closed sequential patterns. Therefore, the COBRA algorithm generates all frequent and only closed sequential patterns.*

5 Discussion

The proposed algorithm, COBRA, is basically a memory-based algorithm, and its efficiency comes from the removal of database scans that is required by BIDE. If the data is too large to fit in the memory space, the partition-and-validation strategy can be used to handle such a case. Therefore, we propose two alternative partition-and-validation strategies to overcome this problem. Details are described below.

- Horizontal-Based Partition (COBRA-HP): Suppose the sequence database is composed of D sequences, we divides the D sequences into k partitions. Each partition can be handled in memory by our algorithms. The local minimum support count for a partition is $minsup$ multiplied by the number of sequences in that partition. To reduce the memory cost, we can apply only $LayerPruning$,

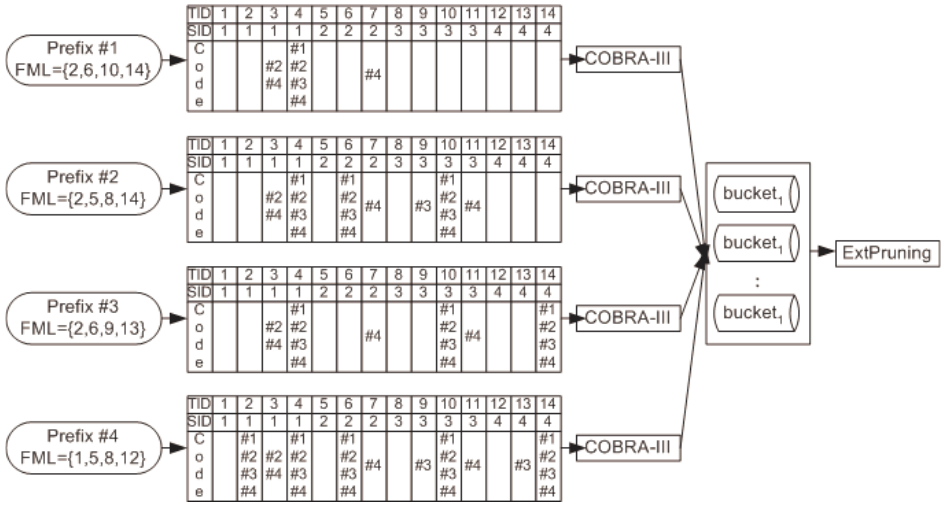


Figure 6. Prefix-Based Partition

output local closed sequences CS of each partition in disk and use $ExtPruning$ to verify CS later. The local closed sequential patterns in disk are false-positive closed frequent sequences. Therefore, an additional validation of the CS is necessary in order to determine the true-positive closed frequent sequences. Finally, we read CS again and remove non-closed sequential patterns. Take Figure 5 as an example, we can divides the 4 sequences into 2 partitions, partitions 1 and 2, as shown in Figure 5. Firstly, we mine the local closed frequent sequences which satisfied the local minimum support in each partition. Finally, we verify CS again to remove non-closed sequences in disk.

- Prefix-Based Partition (COBRA-PP): Different from COBRA-HP, we first run the Phase I and II in COBRA, then store each CFI's FML and EDB in Disk. Therefore, we can load projection database according to CFI's FML to reduce the memory requirement. Next, COBRA Phase III is applied in each partition. Note that we only apply $LayerPruning$ in COBRA Phase III here to minimize the memory cost. COBRA Phase III outputs the potential closed sequential patterns into some disk-based hashing buckets. Finally, the true-positive closed sequences in each buckets are verified by $ExtPruning$. Take Table 1 as an example, we first mine closed frequent itemsets and transform them as EDB. Secondly, we load projection database according to CFI's FML (see Figure 6). Next, we mine closed sequences of each prefix partition and generate potential closed sequential pattern in disk. Finally, $ExtPruning$ is applied to remove non-closed sequences.

Sym.	Definition	Default
D	Number of sequences	10K
C	Average transactions pre sequence	10
T	Average items per transaction	3
N	Number of different items	10K
S	Average transactions in seed sequences	6
I	Average items in seed sequences	3
N_s	Number of maximal potentially large Sequences	2K
N_I	Number of maximal potentially large Itemsets	5K

Figure 7. Parameters for Synthetic Data

6 Experimental Result

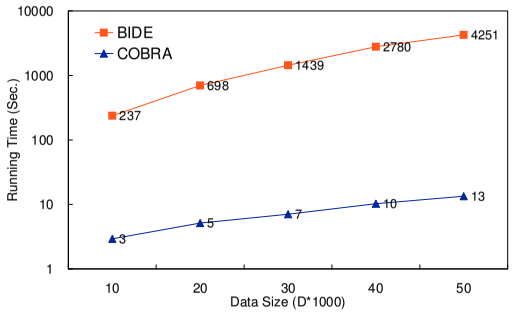
In this section, we report the performance study of the proposed algorithms on both synthetic data and real world data. All the experiments are performed on a 3.2GHz Pentium PC with 3 Gigabytes main memory, running Microsoft Windows XP. All the programs are written in Microsoft/Visual C++ 6.0. In the following experiments, the size of hash table is set to 100.

6.1 Synthetic Data

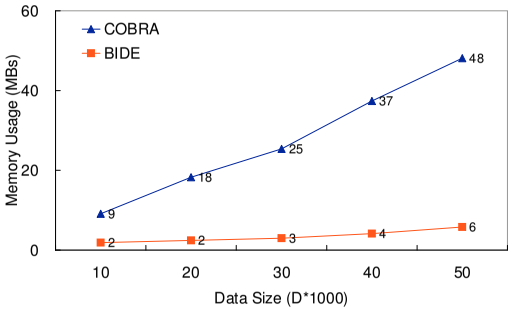
6.1.1 Scalability Test

The synthetic sequence data is generated based on the description in [13]. Table 7 shows the major parameters in this generator and their meanings. We start by looking at the performance of COBRA with default parameter $minsup = 0.5\%$. Figure 8(a) shows the scalability of the algorithms with varying data size. COBRA is two orders of magnitude faster than BIDE for $|D| = 50K$. The scaling of COBRA with database size was linear. Because BIDE needs more scanning time as the data size increases, BIDE has exponential scalability in terms of data size. However, COBRA consumes more memory space than BIDE as shown in Figure 8(b). The main reason is that COBRA maintain the encoded database which are composed of C.F.I.s instead of simple items.

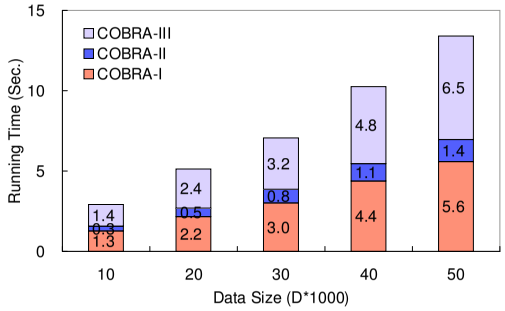
The runtime of COBRA and BIDE on the default data set with varying minimum support threshold, $minsup$, from 0.2% to 0.6% is shown in Figure 8(e). COBRA is faster (90 times) and more scalable than BIDE since the number of sequences checked in the backward extension of BIDE grows rapidly as the $minsup$ decreases, while COBRA only compare the maintained patterns with the newly found pattern. Again, the memory requirement for COBRA increases as $minsup$ decreases since the number of C.F.I.s



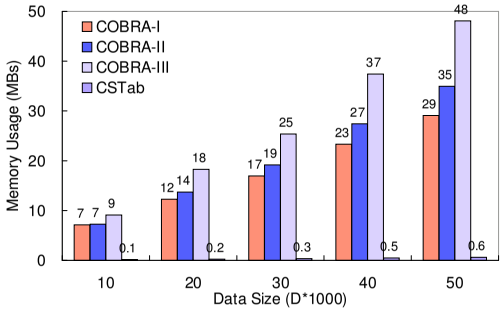
(a) Scaling with Date Size (Time)



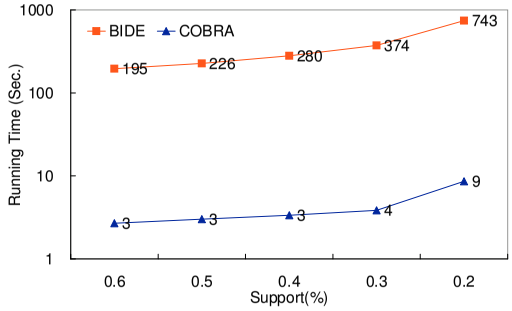
(b) Scaling with Date Size (Space)



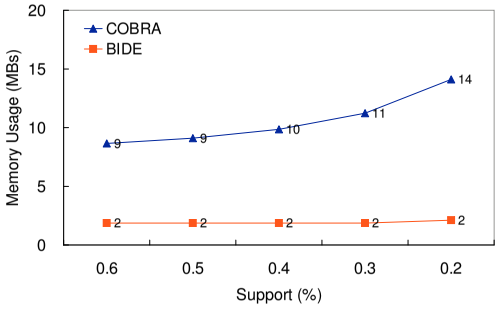
(c) Time cost in each phase of COBRA



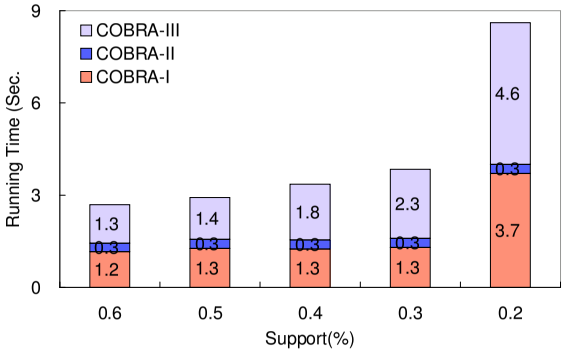
(d) Space cost in each phase of COBRA



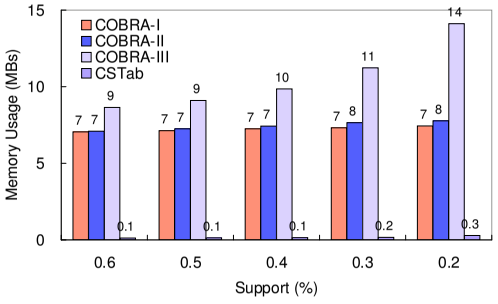
(d) Scaling with $minsup$ (Time)



(f) Scaling with $minsup$ (Space)

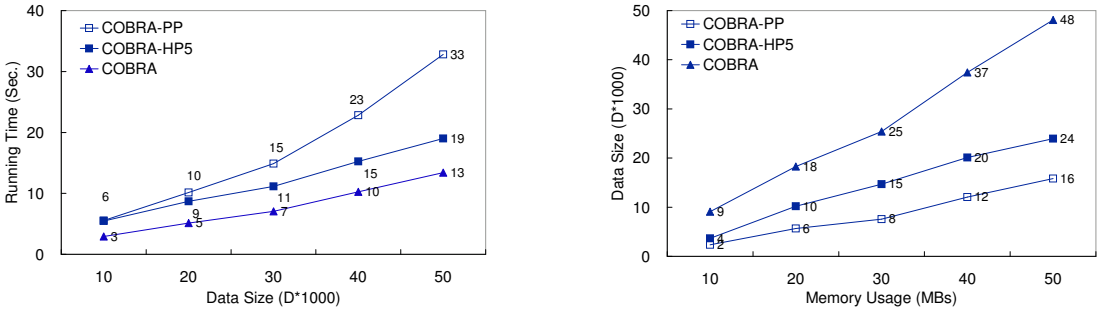


(g) Time cost in each phase of COBRA



(h) Space cost in each phase of COBRA

Figure 8. Scalability Test: Synthetic Data



(a) Partitioning Performance (Time) (b) Partitioning Performance (Space)

Figure 9. Partition-based COBRA: Synthetic Data

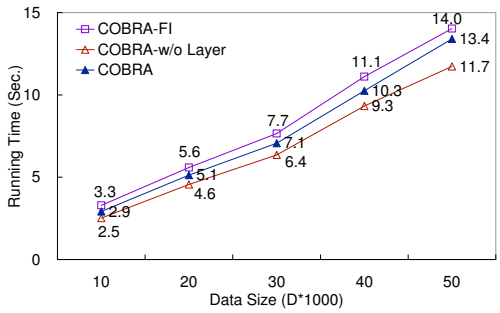
increases as $minsup$ decreases (see Figure 8(f)). In short, the performance study shows that the COBRA algorithm is efficient and scalable for closed sequential pattern mining with acceptable memory cost.

6.1.2 Analysis of Time and Space Cost

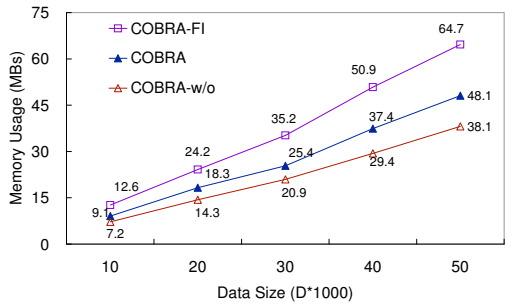
To better understand the algorithm, Figure 8(c)(d)(g)(h) demonstrates the time and space requirement in each phase. Recall that we mine closed frequent itemsets in the first phase, then output C.F.I.s and their $LocationList$ in disk. In the second phase, we load C.F.I. and $LocationList$ from disk to generate the horizontal-based encoded database EDB. Finally, we use the EDB to mine the closed sequential pattern in the third phase. Roughly speaking, the time costs for the three phases are 40%, 10%, and 50%, respectively. As shown in the figure, Phase I (the memory-based CHARM) consumes the most time and space since it maintains the (SID, TID) pairs for each closed frequent itemsets. Due to the nature of phase II, the space requirement for phase II is roughly the size of the encoded database. The space requirement for maintaining closed sequential patterns $CSTab$ (by $ExtPruning$) in phase III is showed in Figure 8(d)(h). Thus, the memory cost in mining process of the Phase III can be estimated by subtracting that of phase III from that of phase II and $CSTab$. For example, the memory cost in the mining process of the Phase III at $minsup = 0.2\%$ is $5.7MB$ $(14MB - 8MB - 0.3MB)$.

6.1.3 Partition-Based COBRA

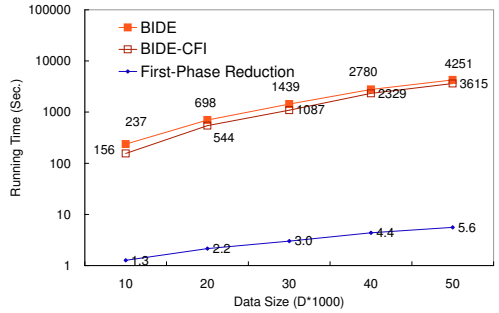
We show the performance of the partition-based COBRA in Figure 9. The experimental result demonstrates that COBRA-PP (Prefix-Based Partition) outperforms COBRA-HP5 (5 Horizontal-Based Partitions) and COBRA in space cost. Since COBRA-PP divides more partitions than COBRA-HP5,



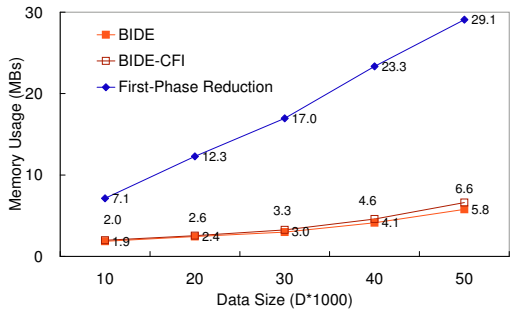
(a) Effectiveness Comparison in Time



(b) Effectiveness Comparison in Space



(c) BIDE v.s. BIDE-CFI in Time



(d) BIDE v.s. BIDE-CFI in Space

Figure 10. The effects of the pruning strategy: Synthetic Data

COBRA-PP needs more time in pattern validation than COBRA-HP5. However, experimental result shows that two alternative partition-and-validation strategies, COBRA-PP and COBRA-HP5, are not only more efficient than BIDE but also reduce the memory requirements of the COBRA.

6.1.4 Effects of the Pruning Strategy

To verify the effectiveness of the first phase reduction and the $LayerPruning$ strategy in the third phase, we demonstrate the experimental results between COBRA, COBRA-FI (COBRA with Frequent Itemset in Phase I) and COBRA-w/o Layer (COBRA without LayerPruning) in Figure 10(a)(b). We can see that COBRA is more effective than COBRA-FI for more codes are generated in the EDB and more closure checking is done by the pruning strategies. As the data size (or minimum support) increases (resp. decreases), the gap between COBRA and COBRA-FI in the running time and space requirement becomes more obvious. This proves the effectiveness of the first-phase reduction. As for the $LayerPruning$ strategy, the effects are case by case depending on the data. Although $LayerPruning$ can remove some search space in the mining process, it also cost a lot of time/space in pattern checking.

Beside, we also use the first-phase reduction in BIDE. Firstly, we mine closed frequent itemsets and

transform them to a encoded database. Next, we use BIDE to mine the closed sequences from encoded database, called BIDE-CFI. Figure 10(c) proves the first-phase reduction can improve the efficiency of the mining process. Although the memory requirements of BIDE-CFI is larger than BIDE (see Figure 10(d)), as the data size increases the gap between BIDE and BIDE-CFI in the running time becomes more substantial.

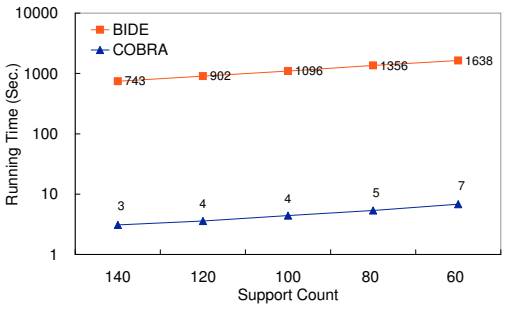
6.2 Real World Data

Next, we run the algorithm on two real world data sets to get a better view of the usefulness of closed sequential patterns. To make the experiments fair to all the algorithms, the real data sets are similar to that used in the performance study in previous works.

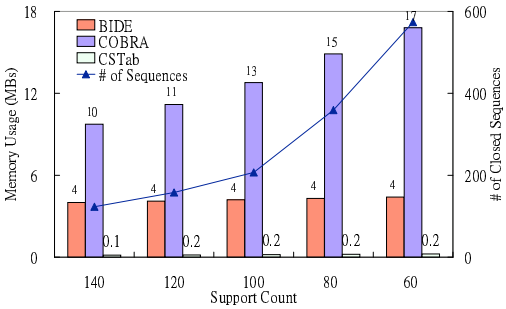
6.2.1 Web Log

The first real data set, $Gazelle$, comes from click-stream data from $http : //gazelle.com$, which was once used in KDD-Cup 2000 competition and is now available through the web site: $http : //www.ecn.purdue.edu/KDDCUP$, more detailed information about this data set can be found in [8]. We use the combination of the productID and AssortmentID as the product $code$. SessionID is considered to identify items in one transaction. For linking itemsets to create a sequence, we use the cookieID. We remove data with unknown '?' productID or AssortmentID. The database contains 27,735 sequences, 33,305 transactions and 70,546 items. There are in total 1,037 distinct items (page views). Note that BIDE only demonstrates this experiments in simple sequences (sequences of item). Similar to CloSpan, we perform this experiments in complex sequences (sequences of itemset).

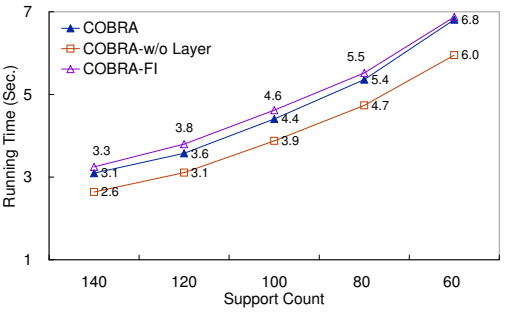
We demonstrate the time and space cost of $Gazelle$ data set in Figure 11 by varying minimum support count from 60 to 140. The memory usage for BIDE is fixed (4MB) while COBRA uses more memory which increases as the support count decreases (from 10 to 17MB). However, the efficiency of COBRA is much better than BIDE in all the cases (by a magnitude of 240). Figure 11(c)(d) also shows the performance of COBRA-FI and COBRA-w/o Layer. The result is similar to that of synthetic data for $Gazelle$ can be viewed as a sparse data set. With the same reason, the first-phase reduction not only enhances the mining speed but also reduces the space requirement. Although $LayerPruning$ remove some redundant search space, this pruning also cost a lot of time/space in pattern validation.



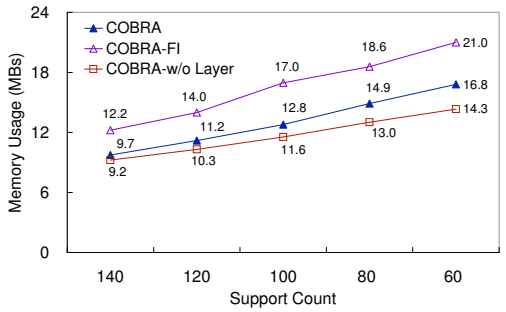
(a) Scalability in Web Log (Time)



(b) Scalability in Web Log (Space)

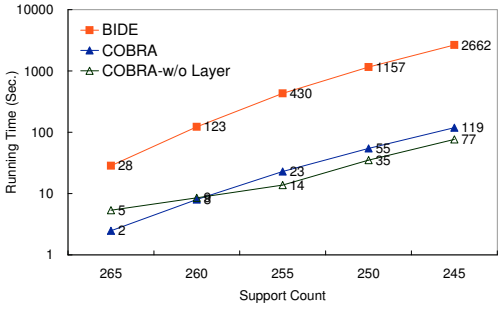


(c) Pruning Strategies in Web Log (Time)

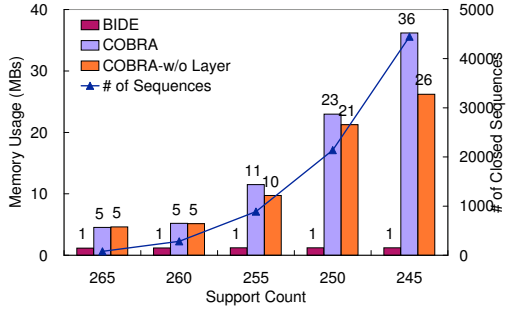


(d) Pruning Strategies in Web Log (Space)

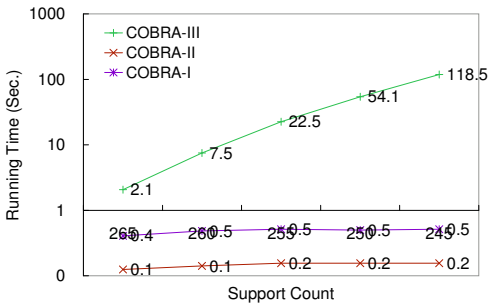
Figure 11. Real World Data: Web Log (Gazelle)



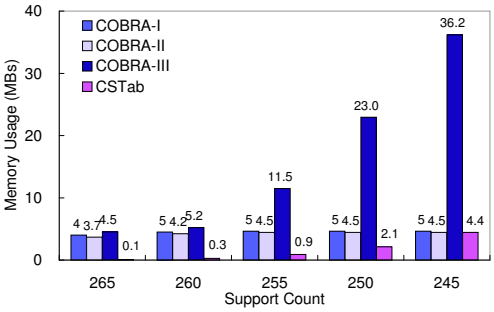
(a) Scalability in Protein Sequence (Time)



(b) Scalability in Protein Sequence (Space)



(c) Time cost in each phase of Protein Sequence



(d) Space cost in each phase of Protein Sequence

Figure 12. Real World Data: Protein (Snake)

6.2.2 Protein Sequence

The second data, $Snake$, is collected from Snake Neurotoxin Database ($http://sdmc.i2r.a-star.edu.sg/Templar/DB/snake\_neurotoxin/$). It contains 272 Toxin-Snake protein sequences which amount to 22,021 items (An average of 81 items for each sequence). $Snake$ is a dense data which contains only 20 distinct items. This data is also used in pattern discovery tasks in many studies [7, 16]. Although BIDE[16] also perform this data in their experiment, their data contain only 175 sequences and it's average length is 67. Note that this data set is composed of simple sequences where each transaction contains one item. Thus, what phase I returns is the set of frequent items. In other words, there's no reduction in the first phase. The comparison will thus show the difference between BIDE and phase III mining.

From Figure 12, we can see that COBRA outperforms BIDE in terms of efficiency, while COBRA consumes more memory for maintaining closed sequences for closure checking by $ExtPruning$. To give a rough figure, BIDE only uses about 1.2MB memory, while COBRA uses about 36.2MB memory to maintain EDB (4.5MB) and 4,448 closed sequences (4.4MB) at support count = 245. Generally speaking, candidate maintenance has such disadvantages that it does not scale well and it costs time in support counting (or database scanning). However, it might potentially prune the search space and gain the odds in time. Thus, it is a tradeoff between time and space. If $ExtPruning$ is implemented at the end of the mining by output the prefix sequences to disk, we can save all the memory but spend more time to complete the closed sequential pattern mining. Even so, the time cost for COBRA is still much less than BIDE. Finally, for this dense data, $LayerPruning$ only improve efficiency at high support (support count $>$ 260).

7 Conclusion

In this paper, we propose a bi-phase reduction approach algorithm for closed sequential pattern mining. Different from previous studies, the mining process is divided into 3-phases: (I) Mining Closed Frequent Itemset; (II) Database Encoding and (III) Mining Closed Sequential Pattern. The proposed algorithm uses both vertical (FML) and horizontal (EDB) database formats to reduce the searching time in the mining process and overcomes some drawbacks in some typical pattern-growth method. Therefore, the proposed algorithm is a memory-based algorithm, and its efficiency comes from the removal of database scans and compressed strategy of bi-phase reduction approach. The experimental results

also demonstrate this approach can be applied to BIDE to better improve the performance of BIDE. Although COBRA consumes more memory space than BIDE, the gain in time cost shows the advantage of COBRA. Besides, memory space cost can be further reduced by partition-and-validation strategies or post (disk-based) $ExtPruning$. Although $LayerPruning$ strategy can remove some search space, it also spend a lot of time and space in pruning validation. Furthermore, how to use the closed sequences in application analysis is also an interesting issue in future work.

References

[1] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'02)*, pages 429–435, 2002.

[2] D. Chiu, Y. Wu, and A. L. P. Chen. An efficient algorithm for mining frequent sequences by a new strategy without support counting. In *Proceedings of the 20th International Conference on Data Engineering (ICDE'04)*, pages 375–386, 2004.

[3] M. N. Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression of constraints. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 14(3):530–552, 2002.

[4] J. Han and J. Pei. Mining frequent patterns by pattern-growth: Methodology and implications. *ACM SIGKDD Explorations (Special Issue on Scalable Data Mining Algorithms)*, 2(2):14–20, 2000.

[5] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu. Freespan: Frequent pattern-projected sequential pattern mining. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00)*, pages 355–359, 2000.

[6] J. Han, J. Wang, Y. Lu, and P. Tzvetkov. Mining top-k frequent closed patterns without minimum support. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, 2002.

[7] I. Jonassen, J. Collins, and D. Higgins. Finding flexible patterns in unaligned protein sequences. *Protein Science*, 4(8):1587–1595, 1995.

[8] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. Kdd-cup 2000 organizers's report: Peeling the onion. *Proceedings of the SIGKDD Explorations*, 2:86–98, 2000.

[9] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings of 7th International Conference on Database Theory (ICDT'99)*, 1999.

[10] J. Pei, G. Dong, W. Zou, and J. Han. On computing condensed frequent pattern bases. In *Proceedings of International Conference on Data Mining (ICDM'02)*, 2002.

[11] J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *Proceedings of the ACM SIGMOD Int. Workshop Data Mining and Knowledge Discovery (SIGMOD'00)*, 2000.

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

[13] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, pages 3–14, 1995.

[14] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology (EDBT'96)*, volume 1057 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 1996.

[15] A. K. Tung, H. Lu, J. Han, and L. Feng. Efficient mining of intertransaction association rules. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(1):43–56, 2003.

[16] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In *Proceedings of the 20th International Conference on Data Engineering (ICDE'04)*, pages 79–90, 2004.

[17] J. Wang, J. Han, and J. Pei. Closet+: Searching for the best strategies for mining. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'03)*, 2003.

[18] X. Yan and R. A. J. Han. Clospan: Mining closed sequential patterns in large datasets. In *Proceedings of the Third SIAM International Conference on Data Mining (SDM)*, 2003.

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

[20] M. J. Zaki and C. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *Proceedings of the 2nd SIAM International Conference on Data Mining (SDM'02)*, 2002.