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

Kuo-Yu Huang, Chia-Hui Chang†, Jiun-Hung Tung and Cheng-Tao Ho

Department of Computer Science and Information Engineering,
National Central University, Chung-Li, Taiwan 320
{want,ginhong,ctho}@db.csie.ncu.edu.tw, †chia@csie.ncu.edu.tw

Abstract. In this work, we study the problem of closed sequential pattern mining. 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. 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 Agrawal et al. [6, 7] introduced the concept of sequential pattern mining in 1995, this problem has received a great deal of attention [2, 12, 1, 5]. 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 [3, 4], 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 [10] and BIDE [9]. 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 least 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 [5] 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 I-step, denoted by $\diamond_i$) or different transaction (sequence extension or S-step, denoted by $\diamond_s$). However this pattern-growth strategy has two drawbacks: duplicate item extensions (To find the closed sequence $<\{A, B\}, \{A, B\}, \{A, B\}>$, we need three item extensions.) and expensive matching cost. 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.

The rest of this paper is organized as follows. We define the problem of closed sequential pattern mining in Section 2. Section 3 presents our algorithm. Experiments are reported in Section 4. Finally, conclusions are made in Section 5.

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.

A sequence database $SD = \{S_1, \ldots, S_{|SD|}\}$ is a set of sequences. Each sequence is associated with a 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.

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.

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	$(C)(A)(D,E)(A,B,C)$

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 a frequent closed itemset if the sequence support is greater than $minsup$ and there exists no superset with the same transaction support.

Example 1. 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 The COBRA Algorithm

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 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 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.*

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 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 1).*

Theorem 1. *A closed sequential pattern is composed of only closed itemsets.*

Proof. 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 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 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 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

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)

Fig. 1. Vertical-based LocationList and FML

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

Fig. 2. Horizontal Encoded Database EDB

only reduces the search spaces and duplicate combinations but also avoids the matching costs in item extension process. A similar framework has also been adopted in [8] 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**.

3.1 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 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 2. 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. *code*s #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 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. By definition (Definition 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 1). Therefore, S_2 is not a closed sequential pattern.

Theorem 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. (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.

Although LayerPruning can prune non-closed sequences during sequence extension step of a prefix sequence, 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 patterns 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 3, while the second rule follows the definition of closed sequential patterns.

3.2 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 Closed Sequential Pattern. Figure 3 shows the pseudo code of the COBRA algorithm. Line 1 calls a modified CHARM [11] 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[11]) 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. 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.

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**,**FML**)) 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**,**FML**)) then return;**
19. **end**
20. **for each** LC_i **in** LC **do**
21. **cobraDFS(**$\alpha \diamond_s LC_i$**,** $LC_i.FML$**);**

Fig. 3. COBRA Algorithm

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 (pNo is chosen to be a prime number. $HSize$ is the size of the hash table.). 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 2 and 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 $\#4.FML <_L \#3.FML$.

$$h(SIDList) = (|SIDList| + \sum_{Sid \in SIDList} Sid * pNo) mod HSize \qquad (1)$$

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 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 remove unnecessary $code$s (line 14). Formally, we define the extended list (EL) and projected database (PDB) of a pattern as follows.

Definition 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 $t'_i = t_i + 1$ and $t'_i.SID = t_i.SID$.*

Definition 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 $\alpha.PDB = \{t'_1, \ldots, t'_m\}$ where $t'_i.SID = t_j.SID$ for some t_j and $t_j < t'_i \leq t_{|SD|}$, where $|SD|$ denotes the number of transactions in the extended databases.*

For example, the projected database for $\alpha = \#2$ (with $\#2.EL = \{3, 6, 9\}$) in Figure 2 is $\#2.PDB = \{3, 4, 6, 7, 9, 10, 11\}$.

Definition 5. *Given a sequence α=$<s_1, \ldots, s_n>$, the **Forward Extended Itemset (FEI)** of α is defined as the set of extended codes 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. The return value of $ExtPruning$ indicates whether the extension of prefix α should go on. If α is a sub-sequence 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. *The COBRA algorithm generates all closed sequential patterns.*

Proof. 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 1, the search space composed by 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 2 and 3). On the other hand, $ExtPruning$ remove only non-closed sequential patterns. Therefore, the COBRA algorithm generates all frequent and only closed sequential patterns.

The proposed algorithm, COBRA, is basically a memory-based algorithm since the number of closed itemsets can be larger than the number of items. 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. Two alternative partition strategies are proposed here: prefix-based partition and horizontal-based partition (see [**?**] for details).

4 Experimental Result

In this section, we report the performance study of the proposed algorithms on synthetic data set. 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.

Scalability Test The synthetic sequence data is generated on the basis of the description in [6]. We start by looking at the performance of COBRA with default parameter $minsup = 0.5\%$. Figure 4(a) shows the scalability of the algorithms with varying data size. COBRA is two orders of magnitude faster than BIDE for 50K sequences. 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 4(b). The main reason is that COBRA maintain the encoded database which is composed by C.F.I.s instead of simple items. For example, COBRA costs approximately 6.6MB for the encoded database maintenance at $|D| = 50K$ and FML costs approximately 10MB.

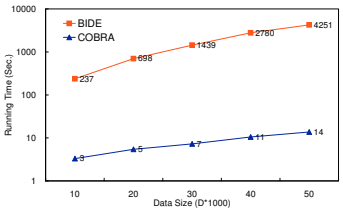
The runtime of COBRA and BIDE on the default data set with varying minimum support threshold, $minsup$, from 0.2% to 0.6% is shown in Figure 4(c). COBRA is faster (90 times) and more scalable than BIDE since the number of sequences checked in the backward extension of BIDE grows rapidly as the $minsup$ 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 increases as $minsup$ decreases (see Figure 4(d)). In short, the performance study shows that the COBRA algorithm is efficient and scalable for closed sequential pattern mining with acceptable memory cost.

To better understand the algorithm, Figure 4(e)(f)(g)(h) demonstrates the time and space expense in each phase. Roughly speaking, the time costs for the three phases are 40%, 10%, and 50%, respectively. As shown in the Figure 4(e)(g), Phase I (the memory-based CHARM) consumes the most time and space since it maintains the (SID, TID) pairs for each closed frequent itemsets. The space requirement for each phase does not vary much since each of them includes both the horizontal encoded database EDB and the vertical database FML. The space requirement for maintaining closed sequential patterns $CSTab$ (by $ExtPruning$) in phase III is also shown in Figure 4(f)(h) for reference.

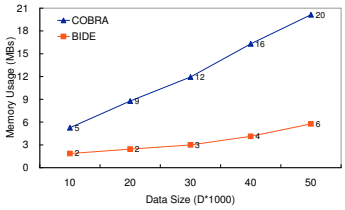
Partition-Based COBRA Figure 5 demonstrates the memory reduction by partition-based COBRA. Prefix-based partition (COBRA-PP) has less memory requirement than horizontal-based partition (COBRA-HP 5 partitions). Since COBRA-PP divides more partitions than COBRA-HP5, COBRA-PP needs more time in pattern validation than COBRA-HP5. However, experimental result shows that both partition-and-validation strategies are not only more efficient than BIDE but also reduce the memory requirements of the COBRA. Thus, while we are trading more space for speed in time, the basic principle is worth trying since the memory cost can be well reduced by partition-based approaches.

5 Conclusion

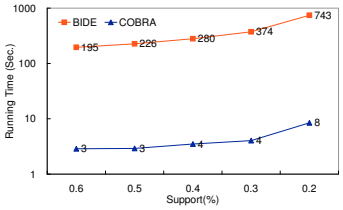
In this paper, we propose a bi-phase reduction approach algorithm for closed sequential pattern mining. Different from previous studies, we first conduct item extension



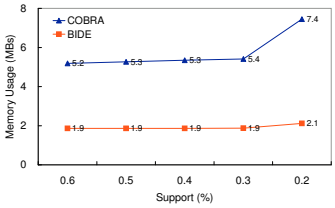
(a) Scaling with Date Size (Time)



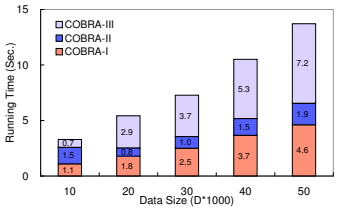
(b) Scaling with Date Size (Space)



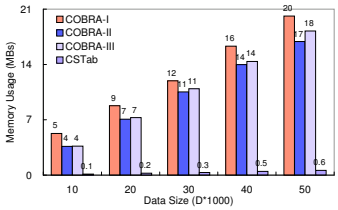
(c) Scaling with $minsup$ (Time)



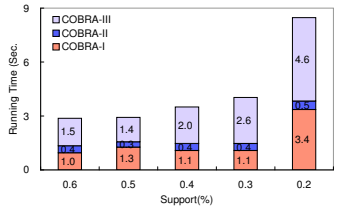
(d) Scaling with $minsup$ (Space)



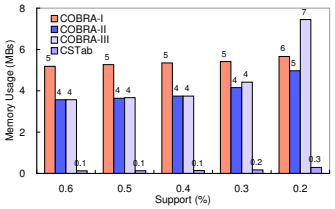
(e) Time cost in each phase of COBRA



(f) Space cost in each phase of COBRA

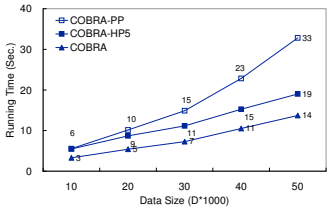


(g) Time cost in each phase of COBRA

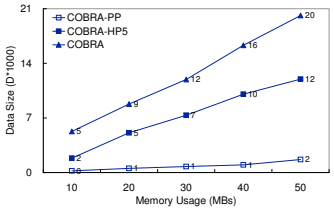


(h) Space cost in each phase of COBRA

Fig. 4. Scalability Test



(a) Partitioning Performance (Time)



(b) Partitioning Performance (Space)

Fig. 5. Partition-based COBRA: Synthetic Data

and then do sequence extension, which overcomes some drawbacks in typical pattern-growth method. 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. Basically, 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. 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) $ExtPruning$.

Acknowledgement

This work was sponsored by National Science Council, Taiwan under grant NSC94-2213-E-008-020.

References

1. 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.
2. 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.
3. 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.
4. J. Pei, G. Dong, W. Zou, and Jiawei Han. On computing condensed frequent pattern bases. In *Proceedings of International Conference on Data Mining (ICDM'02)*, 2002.
5. 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.
6. R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, pages 3–14, 1995.
7. 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.
8. A. K.H. 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.
9. 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.
10. X. Yan and R. Afshar J. Han. Clospan: Mining closed sequential patterns in large datasets. In *Proceedings of the Third SIAM International Conference on Data Mining (SDM)*, 2003.
11. M. J. Zaki and C.J. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *Proceedings of the 2nd SIAM International Conference on Data Mining (SDM'02)*, 2002.
12. M.J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.