

BIDE: Efficient Mining of Frequent Closed Sequences*

Jianyong Wang[†] and Jiawei Han
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801, U.S.A.
{wangj, hanj}@cs.uiuc.edu

Abstract

Previous studies have presented convincing arguments that a frequent pattern mining algorithm should not mine all frequent patterns but only the closed ones because the latter leads to not only more compact yet complete result set but also better efficiency. However, most of the previously developed closed pattern mining algorithms work under the candidate maintenance-and-test paradigm which is inherently costly in both runtime and space usage when the support threshold is low or the patterns become long.

In this paper, we present, BIDE, an efficient algorithm for mining frequent closed sequences without candidate maintenance. It adopts a novel sequence closure checking scheme called BI-Directional Extension, and prunes the search space more deeply compared to the previous algorithms by using the BackScan pruning method and the Scan-Skip optimization technique. A thorough performance study with both sparse and dense real-life data sets has demonstrated that BIDE significantly outperforms the previous algorithms: it consumes order(s) of magnitude less memory and can be more than an order of magnitude faster. It is also linearly scalable in terms of database size.

1 Introduction

Sequential pattern mining, since its introduction in [2], has become an essential data mining task, with broad applications, including market and customer analysis, web log analysis, pattern discovery in protein sequences, and mining XML query access patterns for caching. Efficient mining methods have been studied extensively, including the

*The work was supported in part by National Science Foundation under Grant No. 02-09199, the Univ. of Illinois, and an IBM Faculty Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

[†]Currently is with Digital Technology Center, University of Minnesota at Twin-Cities, email: jianyong@cs.umn.edu.

general sequential pattern mining [13, 20, 9, 24, 17, 4], constraint-based sequential pattern mining [6, 18, 19], frequent episode mining [12], cyclic association rule mining [14], temporal relation mining [5], partial periodic pattern mining [7], and long sequential pattern mining in noisy environment [23].

In recent years many studies have presented convincing arguments that for mining frequent patterns (for both itemsets and sequences), one should not mine *all* frequent patterns but the *closed* ones because the latter leads to not only more compact yet complete result set but also better efficiency [15, 25, 22, 21]. However, unlike mining frequent itemsets, there are not so many methods proposed for mining closed sequential patterns. This is partly due to the complexity of the problem. To our best knowledge, CloSpan is currently the only such algorithm [22]. Like most of the frequent closed itemset mining algorithms, it follows a *candidate maintenance-and-test* paradigm, i.e., it needs to maintain the set of already mined closed sequence candidates which can be used to prune search space and check if a newly found frequent sequence is promising to be closed. Unfortunately, a closed pattern mining algorithm under such a paradigm has rather poor scalability in the number of frequent closed patterns because a large number of frequent closed patterns (or just candidates) will occupy much memory and lead to large search space for the closure checking of new patterns, which is usually the case when the support threshold is low or the patterns become long.

Can we find a way to mine frequent closed sequences without candidate maintenance? This seems to be a very difficult task. In this paper, we present a nice solution which leads to an algorithm, BIDE¹, that mines efficiently the complete set of frequent closed sequences. In BIDE, we do not need to keep track of any single historical frequent closed sequence (or candidate) for a new pattern's closure checking, which leads to our proposal of a deep search space pruning method and some other optimization

¹ BIDE stands for BI-Directional Extension based frequent closed sequence mining.

techniques. Our thorough performance study demonstrates the big success of the algorithm design: BIDE consumes order(s) of magnitude less memory and runs over an order of magnitude faster than the previously developed frequent (closed) sequence mining algorithms, especially when the support is low.

The rest of this paper is organized as follows: In section 2 we present the problem definition of frequent closed sequence mining and discuss the related work and our contributions to this problem. Section 3 is focused on the BIDE algorithm: mainly introducing the *BI-Directional Extension* pattern closure checking mechanism, the *BackScan* pruning method and the *ScanSkip* optimization technique. Some possible extensions are also discussed in this section. In section 4 we present an extensive experimental study. Finally, we conclude the study in section 5.

2 Problem definition and related work

Let $I = \{i_1, i_2, \dots, i_n\}$ be a set of distinct items. A **sequence** S is an ordered list of events, denoted as $\langle e_1, e_2, \dots, e_m \rangle$, where e_i is an item, i.e., $e_i \in I$ for $1 \leq i \leq m$. For brevity, a sequence is also written as $e_1 e_2 \dots e_m$. From the definition we know that an item can occur multiple times in different events of a sequence. The number of events (i.e., instances of items) in a sequence is called the length of the sequence and a sequence with a length l is also called an l -sequence. For example, $AABCCA$ is a 6-sequence. A sequence $S_a = a_1 a_2 \dots a_n$ is *contained* in another sequence $S_b = b_1 b_2 \dots b_m$, if there exist integers $1 \leq i_1 < i_2 < \dots < i_n \leq m$ such that $a_1 = b_{i_1}, a_2 = b_{i_2}, \dots, a_n = b_{i_n}$. If sequence S_a is contained in sequence S_b , S_a is called a **subsequence** of S_b and S_b a **supersequence** of S_a , denoted as $S_a \sqsubseteq S_b$.

An input sequence database SDB is a set of tuples (sid, S) , where sid is a sequence identifier, and S an input sequence. The number of tuples in SDB is called the **base size** of SDB , denoted as $|SDB|$. A tuple (sid, S) is said to *contain* a sequence S_α , if S is a supersequence of S_α , i.e., $S_\alpha \sqsubseteq S$. The **absolute support** of a sequence S_α in a sequence database SDB is the number of tuples in SDB that *contain* S_α , denoted as $sup^{SDB}(S_\alpha)$, and the **relative support** is the percentage of tuples in SDB that *contain* S_α (i.e., $sup^{SDB}(S_\alpha)/|SDB|$). Without loss of generality, we use the absolute support for describing the BIDE algorithm while using the relative support to present the experimental results in the remaining of the paper.

Given a support threshold min_sup , a sequence S_α is a **frequent sequence** on SDB if $sup^{SDB}(S_\alpha) \geq min_sup$. If sequence S_α is frequent and there exists no proper supersequence of S_α with the same support, i.e., $\nexists S_\beta$ such that $S_\alpha \sqsubset S_\beta$ and $sup^{SDB}(S_\alpha) = sup^{SDB}(S_\beta)$, we call S_α a **frequent closed sequence**. The problem of mining

Sequence identifier	Sequence
1	C A A B C
2	A B C B
3	C A B C
4	A B B C A

Table 1. An example sequence database SDB.

frequent closed sequences is to find the complete set of frequent closed sequences for an input sequence database SDB , given a minimum support threshold, min_sup .

Example 1 Table 1 shows the input sequence database SDB in our running example. The database has totally 3 unique items, four input sequences (i.e., $|SDB|=4$). Suppose $min_sup = 2$. The complete set of frequent closed sequences, $S_{fcs} = \{AA:2, ABB:2, ABC:4, CA:3, CABC:2, CB:3\}$, consists of only six sequences, while the whole set of frequent sequences consists of 17 sequences, that is, $S_{fs} = \{A:4, AA:2, AB:4, ABB:2, ABC:4, AC:4, B:4, BB:2, BC:4, C:4, CA:3, CAB:2, CABC:2, CAC:2, CB:3, CBC:2, CC:2\}$. Obviously, S_{fcs} is more compact than S_{fs} . Also, if a frequent sequence, S_α , has the same support as that of one of its proper supersequence, S_β , S_α is *absorbed* by S_β . For example, frequent sequence $CBC:2$ is absorbed by sequence $CABC:2$, because $(CBC \sqsubset CABC)$ and $(sup^{SDB}(CBC) = sup^{SDB}(CABC) = 2)$. ■

Notice that in the above definition of a sequence, each event contains only a single item. Thus the derived BIDE algorithm mines only frequent closed *single-item* sequences. In section 3.6, we show that BIDE can be easily extended to mine closed sequences of subsets of items (e.g., sequences of shopping transactions). We first discuss mining single-item sequences because (1) it makes the presentation clear by focusing on the methodology and optimization techniques instead of tedious description, and (2) it represents one of the most important and popular type of sequences, such as DNA strings, protein sequences, Web click streams, and sequences of file block references in operating systems.

2.1 Related work

The sequential pattern mining problem was first proposed by Agrawal and Srikant in [2], and the same authors further developed a generalized and refined algorithm, GSP [20], based on the Apriori property [1]. Since then, many sequential pattern mining algorithms have also been proposed for performance improvements. Among those, SPADE [24], PrefixSpan [17], and SPAM [4] are quite interesting ones. SPADE is based on a vertical id-list format and uses a lattice-theoretic approach to decompose the original search space into smaller spaces, while PrefixSpan adopts

a horizontal format dataset representation and mines the sequential patterns under the pattern-growth paradigm: grow a prefix pattern to get longer sequential patterns by building and scanning its projected database. Both SPADE and PrefixSpan outperform GSP. SPAM is a recently developed algorithm for mining long sequential patterns and adopts a vertical bitmap representation. Its performance study shows that SPAM is more efficient in mining long patterns than SPADE and PrefixSpan, however, it consumes more space in comparison with SPADE and PrefixSpan.

Since the introduction of frequent closed itemset mining [15], several efficient frequent closed itemset mining algorithms have been proposed, such as A-Close [15], CLOSET [16], CHARM [25], and CLOSET+ [21]. Most of these algorithms need to maintain the already mined frequent closed patterns in order to do pattern closure checking. To reduce the memory usage and search space for pattern closure checking, two algorithms, TFP [8] and CLOSET+², adopt a compact 2-level hash indexed result-tree structure to store the already mined frequent closed itemset candidates. Some of the pruning methods and pattern closure checking schemes proposed there can be extended for optimizing the mining of closed sequential patterns as well.

CloSpan is a recently proposed algorithm for mining frequent closed sequences [22]. It follows the *candidate maintenance-and-test* approach: First generate a set of closed sequence candidates which is stored in a hash-indexed result-tree structure and then do post-pruning on it. It uses some pruning methods like *CommonPrefix* and *Backward Sub-Pattern pruning* to prune the search space. Because CloSpan needs to maintain the set of historical closed sequence candidates, it will consume much memory and lead to huge search space for pattern closure checking when there are many frequent closed sequences. As a result, it does not scale very well with respect to the number of frequent closed sequences.

Contributions. In this paper, we introduce BIDE, an efficient algorithm for discovering the complete set of frequent closed sequences. The contributions of this paper include: (1) A new paradigm is proposed for mining closed sequences without candidate maintenance, called *BI-Directional Extension*. The forward directional extension is used to grow the prefix patterns and also check the closure of prefix patterns, while the backward directional extension can be used to both check closure of a prefix pattern and prune the search space. (2) Under the BI-Directional Extension paradigm, we designed an efficient algorithm for frequent closed sequence mining, BIDE. The *BI-Directional Extension* pattern closure checking scheme, the *BackScan*

² CLOSET+ adopts a hybrid closure checking scheme: the *result tree* method for dense datasets and *upward checking* for sparse datasets, among which the *upward checking* can be regarded as a simplified version of the *backward-extension event checking* described in Lemma 2 of this paper.

pruning method, and the *ScanSkip* optimization technique are proposed to speed up the mining and also assure the correctness of the algorithm. (3) Our thorough performance study shows that BIDE has surprisingly high efficiency: it can be an order of magnitude faster than CloSpan but only uses order(s) of magnitude less memory in many cases. It also has very good scalability w.r.t. the database size.

3 BIDE: An efficient algorithm for frequent closed sequence mining

In this section, we introduce the BIDE algorithm by answering the following questions: How to enumerate the complete set of frequent sequences? Upon getting a frequent sequence, how to check if it is closed? How to design some search space pruning methods or other optimization techniques to accelerate the mining process?

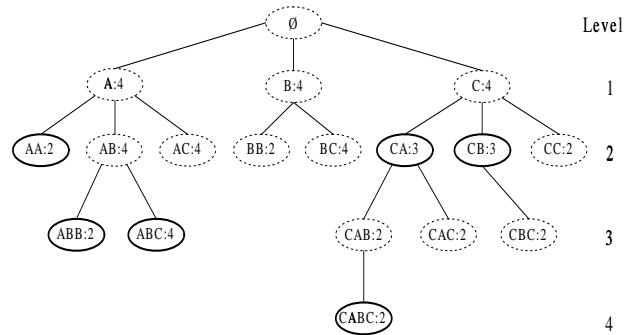


Fig. 1. The lexicographic frequent sequence tree in our running example.

3.1 Frequent sequence enumeration

Assume there is a lexicographical ordering \leq among the set of items I in the input sequence database (e.g., in our running example, one possible item ordering can be $A \leq B \leq C$), conceptually the complete search space of sequence mining forms a sequence tree [4], which can be constructed in the following way: The root node of the tree is at the top level and labeled with \emptyset , recursively we can extend a node N at level L in the tree by adding one item in I to get a child node at the next level $L+1$ and the children of a node N are generated and arranged according to the chosen lexicographical ordering. By removing the infrequent sequences in the sequence tree, the remaining nodes in the lattice form a lexicographic frequent sequence tree, which contains the complete set of frequent sequences. Fig. 1 shows the lexicographic frequent sequence tree built from our running example. In Fig. 1, each node contains a frequent sequence and its corresponding support, and the sequences in the dotted ellipses are non-closed ones.

Many previous frequent pattern (either itemset or sequence) mining algorithms have elaborated that depth-first searching is more efficient in mining long patterns than breadth-first searching [4]. BIDE traverses the sequence tree in a strict depth-first search order. In our example shown in Fig. 1, the frequent sequences will be mined and reported in such an order: $A:4, AA:2, AB:4, ABB:2, ABC:4, AC:4, B:4, BB:2, BC:4, C:4, CA:3, CAB:2, CAB:2, CAC:2, CB:3, CBC:2, CC:2$.

A certain node in the sequence tree can be treated as a prefix sequence, from which the set of its children can be generated by adding one item in I . Some items may not be locally frequent with respect to (abbreviated w.r.t.) the corresponding prefix sequence. Because we are only interested in mining frequent sequences, according to the *downward closure property* (also called the *Apriori property* [1]), we only need to grow a prefix sequence using the set of its locally frequent items. To compute the locally frequent items w.r.t. a certain prefix, a well-known method is to build the projected database for the prefix and scan it to count the items. Two kinds of projection methods have been used in the past: *physical projection* and *pseudo projection* [17]. Because the physical projection-based method needs to physically build the conditional projected databases, it is not space- and runtime- efficient due to the cost of allocating and freeing memory. In BIDE, we only use the pseudo projection method to find the set of locally frequent items w.r.t. a certain prefix and use them to grow the corresponding prefix. Here we briefly introduce the pseudo-projection method (for details, see [17]).

Definition 1 (First instance of a prefix sequence) Given an input sequence S which contains a prefix 1-sequence e_1 , the subsequence from the beginning of S to the first appearance of item e_1 in S is called the first instance of prefix 1-sequence e_1 in S . Recursively, we can define the first instance of a $(i + 1)$ -sequence $e_1e_2 \dots e_i e_{i+1}$ from the first instance of the i -sequence $e_1e_2 \dots e_i$ (where $i \geq 1$) as the subsequence from the beginning of S to the first appearance of item e_{i+1} which also occurs after the first instance of the i -sequence $e_1e_2 \dots e_i$. For example, the first instance of the prefix sequence AB in sequence $CAABC$ is $CAAB$. ■

Definition 2 (Projected sequence of a prefix sequence) Given an input sequence S which contains a prefix i -sequence $e_1e_2 \dots e_i$, the remaining part of S after we remove the first instance of the prefix i -sequence $e_1e_2 \dots e_i$ in S is called the projected sequence w.r.t. prefix $e_1e_2 \dots e_i$ in S . For example, the projected sequence of prefix sequence AB in sequence $ABBCA$ is BCA . ■

Definition 3 (Projected database of a prefix sequence) Given an input sequence database SDB , the complete set of projected sequences in SDB w.r.t. a prefix sequence

$e_1e_2 \dots e_i$ is called the projected database w.r.t. prefix $e_1e_2 \dots e_i$ in SDB . For example, the projected database of prefix sequence AB in our running example is $\{C, CB, C, BCA\}$. ■

After giving the definition of the projected database for a certain prefix sequence, the idea of pseudo-projection can be described as follows. Instead of physically constructing the projected database, we only need to record a set of pointers, one for each projected sequence, pointing at the starting position in the corresponding projected sequence. By following the set of pointers, it is easy to locate the set of projected sequences. And by scanning forward each projected sequence w.r.t. a prefix S_p and count the items (This is the so-called *Forward-extension* step), we will find the locally frequent items w.r.t. prefix S_p , which can be used to grow prefix S_p in order to get longer frequent prefix sequences. For example, if $S_p=AB$, the set of its locally frequent items is $\{C:4, B:2\}$.

<p>Frequent-sequence-enumeration (SDB, min_sup, FS) Input: an input sequence database SDB, a minimum support threshold min_sup Output: the complete set of frequent sequences, FS 1: $FS = \emptyset$; 2: call <i>Frequent-sequences</i>($SDB, \emptyset, min_sup, FS$); 3: return FS;</p> <p>Frequent-sequences ($S_p_SDB, S_p, min_sup, FS$) Input: a projected sequence database S_p_SDB, a prefix sequence S_p, and a minimum support threshold min_sup Output: the current set of frequent sequences, FS 4: if S_p is non-empty 5: $FS = FS \cup S_p$; 6: $LF_S_p =$ locally frequent items (S_p_SDB, S_p, min_sup); 7: if LF_S_p is empty 8: Return; 9: for each locally frequent item i 10: $S_p^i = \langle S_p, i \rangle$; 11: $S_p^i_SDB =$ pseudo projected database (S_p^i, S_p_SDB); 12: call <i>Frequent-sequences</i>($S_p^i_SDB, S_p^i, min_sup, FS$);</p>

Fig. 2. Frequent sequence enumeration algorithm.

Fig. 2 shows the algorithm to enumerate the complete set of frequent sequences, which is similar to the pseudo-projection-based PrefixSpan algorithm. It recursively calls subroutine *Frequent-sequences* ($S_p_SDB, S_p, min_sup, FS$): For a certain prefix S_p , if it is non-empty, output it (line 4 and 5), scan projected database S_p_SDB once to find the locally frequent items (line 6), each frequent item i can be chosen in lexicographical ordering to grow S_p to get a new prefix S_p^i (line 10), scan S_p_SDB once again to build pseudo-projection database for each new prefix S_p^i (line 11). Furthermore, one can easily figure out that the order of the frequent sequence enumeration is consistent with

the depth-first traversal of the frequent sequence tree.

3.2 The *BI-Directional Extension closure checking scheme*

The frequent enumeration algorithm in Fig. 2 can only be used to mine the complete set of frequent sequences instead of the frequent closed ones. Usually upon getting a new frequent prefix sequence, we need to do *pattern closure checking* in order to assure that it is really closed, i.e., it cannot be *absorbed* by one of its super-sequences with the same support. Currently most of the frequent closed pattern (both itemset and sequence) mining algorithms, like CLOSET [16], CHARM [25], TFP [8] and CloSpan [22], need to maintain the set of already mined frequent closed patterns (or just candidates) in memory and do (1) *subpattern checking*, that checks if a newly found pattern can be absorbed by an already mined frequent closed pattern (or candidate); and/or (2) *super-pattern checking*, which checks whether the newly found pattern can absorb some already mined closed pattern candidates.

For a properly designed closed itemset mining algorithm like CHARM, it only needs to do the subpattern checking, which is usually less expensive than super-pattern checking. However, a typical frequent closed sequence mining algorithm usually needs to do both subpattern and super-pattern checking. Here we can use our running example to explain it. Let the lexicographical ordering be $B \leq A \leq C$. Upon getting a new frequent sequence $ABC:4$, another frequent sequence $BC:4$ has already been mined, which can be absorbed by $ABC:4$. Therefore, we need to do super-pattern checking in order to remove the previously mined but non-closed frequent sequences. In addition, when we get a new prefix sequence $C:4$, another frequent sequence $ABC:4$ has already been mined, which can absorb $C:4$. As a result, we also need to do sub-pattern checking in order to remove the newly found but non-closed sequence.

It is easy to see that because the above pattern closure checking scheme adopted by the previous algorithms needs to maintain the already mined frequent closed patterns (or candidates) in memory, the algorithms like CloSpan may consume much memory and the search space for pattern closure checking will be huge when there exist a large number of frequent closed sequences. Some closed itemset mining algorithms such as TFP [8] try to save space by storing the closed itemset candidates in a compact prefix itemset-tree structure and reduce the search space by applying a two-level hash-index. CloSpan adopts the similar techniques, however, because a prefix sequence tree is usually less compact than an itemset tree, it still consumes much memory.

To avoid maintaining the set of already mined closed sequence candidates in memory, we have designed a new

sequence closure checking scheme, called *BI-Directional Extension checking*. According to the definition of a frequent closed sequence, if an n -sequence, $S=e_1e_2\dots e_n$, is non-closed, there must exist at least one event, e' , which can be used to extend sequence S to get a new sequence, S' , which has the same support. The sequence S can be extended in three ways: (1) $S' = e_1e_2\dots e_n e'$ and $sup^{SDB}(S') = sup^{SDB}(S)$; (2) $\exists i (1 \leq i < n)$, $S' = e_1e_2\dots e_i e' e_{i+1} \dots e_n$ and $sup^{SDB}(S') = sup^{SDB}(S)$; and (3) $S' = e' e_1 e_2 \dots e_n$ and $sup^{SDB}(S') = sup^{SDB}(S)$. In the first case, event e' occurs after event e_n , we call e' a *forward-extension event* (or item) and S' a *forward-extension sequence* w.r.t. S . While in the second and third cases, event e' occurs before event e_n , we call e' a *backward-extension event* (or item) and S' a *backward-extension sequence* w.r.t. S . After giving the above definition, the following theorem will be evident according to the definition of a frequent closed sequence.

Theorem 1 (*BI-Directional Extension closure checking*) *If there exists no forward-extension event nor backward-extension event w.r.t. a prefix sequence S_p , S_p must be a closed sequence; otherwise, S_p must be non-closed. ■*

From theorem 1 we know that to judge if a frequent prefix sequence is closed, we need to check whether there is any forward-extension event or backward-extension event. It is relatively easy to find the forward-extension events according to the following Lemma.

Lemma 1 (*Forward-extension event checking*) *For a prefix sequence S_p , its complete set of forward-extension events is equivalent to the set of its locally frequent items whose supports are equal to $SUP^{SDB}(S_p)$.*

Proof. *The locally frequent items are found by scanning the projected database w.r.t. S_p , which consists of all the projected sequences. Since each event in a projected sequence always occurs after the prefix sequence S_p , if it occurs in every projected sequence, it forms a forward-extension event. Also, any event occurring after the first instance of S_p must be included in the projected database, which means the complete set of forward-extension events can be found by scanning the projected database w.r.t. S_p . ■*

Definition 4 (*Last instance of a prefix sequence*) Given an input sequence S which contains a prefix i -sequence $e_1e_2\dots e_i$, the last instance of the prefix sequence $e_1e_2\dots e_i$ in S is the subsequence from the beginning of S to the last appearance of item e_i in S . For example, the last instance of the prefix sequence AB in sequence $ABBCA$ is ABB . ■

Definition 5 (*The i -th last-in-last appearance w.r.t. a prefix sequence*) For an input sequence S containing a prefix

n -sequence, $S_p=e_1e_2\dots e_n$, the i -th last-in-last appearance w.r.t. the prefix S_p in S is denoted as LL_i and defined recursively as: (1) if $i = n$, it is the last appearance of e_i in the last instance of the prefix S_p in S ; (2) if $1 \leq i < n$, it is the last appearance of e_i in the last instance of the prefix S_p in S while LL_i must appear before LL_{i+1} . For example, if $S=CAABC$ and $S_p=AB$, the 1st last-in-last appearance w.r.t. prefix S_p in S is the second A in S . ■

Definition 6 (The i -th maximum period of a prefix sequence) For an input sequence S containing a prefix n -sequence $S_p=e_1e_2\dots e_n$, the i -th maximum period of the prefix S_p in S is defined as: (1) if $1 < i \leq n$, it is the piece of sequence between the end of the first instance of prefix $e_1e_2\dots e_{i-1}$ in S and the i -th last-in-last appearance w.r.t. prefix S_p ; (2) if $i = 1$, it is the piece of sequence in S locating before the 1st last-in-last appearance w.r.t. prefix S_p . For example, if $S=ABCB$ and the prefix sequence $S_p=AB$, the 2nd maximum period of prefix S_p in S is BC , while the 1st maximum period of prefix S_p in S is \emptyset . ■

Lemma 2 (Backward-extension event checking) Let the prefix sequence be a n -sequence, $S_p=e_1e_2\dots e_n$. If $\exists i$ ($1 \leq i \leq n$) and there exists an item e' which appears in each of the i -th maximum periods of the prefix S_p in SDB , e' is a backward-extension event (or item) w.r.t. prefix S_p . Otherwise, for any i ($1 \leq i \leq n$), if we cannot find any item which appears in each of the i -th maximum periods of the prefix S_p in SDB , there will be no backward-extension event w.r.t. prefix S_p .

Proof. From the definition of the i -th maximum period of a prefix sequence, we know if item e' appears in each of the i -th maximum periods of the prefix sequence S_p , we can get a new sequence $S'_p=e_1e_2\dots e_{i-1}e'e_i\dots e_n$ ($1 < i \leq n$) or $S'_p=e'e_1e_2\dots e_n$ ($i = 1$), which satisfies $S'_p \supset S_p$ and $\text{sup}^{SDB}(S'_p) = \text{sup}^{SDB}(S_p)$. Therefore, e' is a backward-extension item w.r.t. prefix S_p and S_p is not closed.

In addition, assume there exists a sequence $S'_p=e'e_1e_2\dots e_n$ (for $i = 1$) or $S'_p=e_1e_2\dots e_{i-1}e'e_i\dots e_n$ (for $1 < i \leq n$), which can absorb S_p , that is, item e' is a backward-extension item w.r.t. S_p . In each sequence containing S_p , item e' must appear after the first instance of prefix $e_1e_2\dots e_{i-1}$ (for $1 < i \leq n$) and before the i -th last-in-last appearance w.r.t. S_p , which means item e' must appear in the i -th maximum period of S_p . As a result, for any i ($1 \leq i \leq n$), if we cannot find any item which appears in each of the i -th maximum periods of the prefix S_p in SDB , there will be no backward-extension event w.r.t. prefix S_p . ■

We use an example to illustrate the sequence closure checking scheme in BIDE. First, we assume $S_p=AC:4$, it is easy to find that item B appears in each of the 2nd maximum periods of S_p . As a result $AC:4$ is not closed. In

contrast, let $S_p=ABC:4$, we cannot find any backward-extension item for it. Also there is no forward-extension item for it, therefore $ABC:4$ is a frequent closed sequence.

3.3 The BackScan search space pruning method

Upon finding a new frequent sequence by the frequent sequence enumeration algorithm, we can use the above *BI-Directional Extension* closure checking scheme to check if it is closed in order to generate the complete set of non-redundant frequent sequences. Although the closure checking scheme can lead to more compact result set, it cannot improve the mining efficiency. As Fig. 1 shows the whole subtree under node ' $B:4$ ' contains no frequent closed sequences, which means there is no hope to grow prefix $B:4$ to generate any frequent closed sequences. If we can detect such unpromising prefix sequences and stop growing them, the search space will be reduced.

Search space pruning in frequent closed sequence mining is trickier than that in frequent closed itemset mining. Usually a depth-first search based closed itemset mining algorithm like CLOSET can stop growing a prefix itemset once it finds that this itemset can be absorbed by an already mined closed itemset. However, a closed sequence mining algorithm cannot do so. For example, assume the lexicographical ordering in our running example is $A \leq B \leq C$, and the current prefix sequence is $C:4$, which can be absorbed by an already mined sequence $ABC:4$, but we cannot stop growing $C:4$. As Fig. 1 shows, we can still generate three frequent closed sequences (i.e., $CA:3$, $CABC:2$, and $CB:3$) by growing prefix $C:4$. This complicated situation is caused by the multiple instances of the same item in a sequence and the temporal ordering among the events in a sequence.

Definition 7 (The i -th last-in-first appearance w.r.t. a prefix sequence) For an input sequence S containing a prefix n -sequence $S_p=e_1e_2\dots e_n$, the i -th last-in-first appearance w.r.t. the prefix S_p in S is denoted as LF_i and defined recursively as: (1) if $i = n$, it is the last appearance of e_i in the first instance of the prefix S_p in S ; (2) if $1 \leq i < n$, it is the last appearance of e_i in the first instance of the prefix S_p in S while LF_i must appear before LF_{i+1} . For example, if $S=CAABC$ and $S_p=CA$, the 2nd last-in-first appearance w.r.t. prefix S_p in S is the first A in S . ■

Definition 8 (The i -th semi-maximum period of a prefix sequence) For an input sequence S containing a prefix n -sequence $S_p=e_1e_2\dots e_n$, the i -th semi-maximum period of the prefix S_p in S is defined as: (1) if $1 < i \leq n$, it is the piece of sequence between the end of the first instance of prefix $e_1e_2\dots e_{i-1}$ in S and the i -th last-in-first appearance w.r.t. prefix S_p ; (2) if $i = 1$, it is the piece of sequence in

S locating before the 1st last-in-first appearance w.r.t. prefix S_p . For example, if $S=ABCB$ and the prefix sequence $S_p=AC$, the 2nd semi-maximum period of prefix AC in S is B , while the 1st semi-maximum period of prefix AC in S is \emptyset . ■

Theorem 2 (BackScan search space pruning) *Let the prefix sequence be an n -sequence, $S_p=e_1e_2\dots e_n$. If $\exists i$ ($1 \leq i \leq n$) and there exists an item e' which appears in each of the i -th semi-maximum periods of the prefix S_p in SDB , we can safely stop growing prefix S_p .*

Proof. *Because item e' appears in each of the i -th semi-maximum periods of the prefix S_p in SDB , we can get a new prefix $S'_p=e_1e_2\dots e_{i-1}e'e_i\dots e_n$ ($1 < i \leq n$) or $S'_p=e'e_1e_2\dots e_n$ ($i = 1$), and both $(S_p \sqsubset S'_p)$ and $(sup^{SDB}(S_p) = sup^{SDB}(S'_p))$ hold. Any locally frequent item e'' w.r.t. prefix S_p is also a locally frequent item w.r.t. S'_p , in the meantime $(\langle S_p, e'' \rangle \sqsubset \langle S'_p, e'' \rangle)$ and $(sup^{SDB}(\langle S_p, e'' \rangle) = sup^{SDB}(\langle S'_p, e'' \rangle))$ hold. This means there is no hope to mine frequent closed sequences with prefix S_p . ■*

For example, if prefix sequence $S_p=B:4$, there is an item A which appears in each of the 1st semi-maximum period of prefix S_p in SDB , we can safely stop mining frequent closed sequences with prefix $B:4$. In contrast, if $S_p=C:4$, we cannot find any item which appears in each of the 1st semi-maximum periods of prefix S_p in SDB . As a result, we cannot stop growing $C:4$.

Compared with some pruning methods used in previously developed algorithms [16, 25, 22], which are based on the relationships among the newly found frequent pattern and some already mined closed patterns (or just candidates), the *BackScan* pruning method is more aggressive and thus more effective. Consider another possible lexicographical ordering in our running example $B \leq A \leq C$, which means we first mine the closed sequences with prefix B . According to Theorem 2, we can safely prune prefix B and directly mine frequent closed sequences with prefix A . However, because there are no other already mined frequent closed sequences (or candidates) for checking, algorithms based on the *candidate-maintenance-and-test* paradigm will still try to use B as a prefix to grow frequent closed sequences.

3.4 The *ScanSkip* optimization technique

The above closure checking scheme needs to scan backward a set of maximum-periods w.r.t. a certain prefix, this is one of the most expensive operations in BIDE. Assume the current prefix sequence is $S_p=e_1e_2\dots e_n$ with a support SUP_{S_p} . For any i ($1 \leq i \leq n$), let $\{MP_1^i, MP_2^i, \dots, MP_{SUP_{S_p}}^i\}$ be the SUP_{S_p} i -th maximum periods w.r.t. S_p . By scanning the k -th i -th ($1 \leq i \leq n$) maximum period we

can get a set of unique items, denoted as SI_k^i . The set of items which appears in each of the i -th maximum periods w.r.t. S_p equals $SI_1^i \cap SI_2^i \cap \dots \cap SI_{SUP_{S_p}}^i$ and is denoted as \cap_{SI^i} .

In the sequence closure checking scheme, we only care about whether \cap_{SI^i} is empty or not. If we find that after we scan the first m i -th maximum periods, the intersection of the set of items which appears in each of the first m i -th maximum periods has become empty, we will do not need to scan the left ($SUP_{S_p}-m$) i -th maximum periods, because we already know \cap_{SI^i} must be empty. We call this optimization the *ScanSkip* technique. Similarly, in the *BackScan* search space pruning method, BIDE needs to scan backward a number of semi-maximum periods w.r.t. a prefix sequence. The *ScanSkip* technique can also be used to speed up the *BackScan* search space pruning.

Here we use an example to illustrate the usefulness of the *ScanSkip* optimization technique. In our running example shown in Table 1, assume the current prefix sequence is $ABC:4$. The set of the 3rd maximum periods w.r.t. $ABC:4$ is $\{\emptyset, \emptyset, \emptyset, B\}$, after scanning the first 3rd maximum period we find that it contains no item, we know that there will be no item which appears in each of the four 3rd maximum periods w.r.t. prefix $ABC:4$ without scanning the last three 3rd maximum periods. The set of the 2nd maximum periods w.r.t. $ABC:4$ is $\{A, \emptyset, \emptyset, B\}$, after scanning the first two 2nd maximum periods, we already know that there will be no item which appears in each of the four 2nd maximum periods w.r.t. prefix $ABC:4$. Similarly, The set of the 1st maximum periods w.r.t. $ABC:4$ is $\{CA, \emptyset, C, \emptyset\}$, we can skip the scanning of the last two 1st maximum periods w.r.t. $ABC:4$.

3.5 The BIDE algorithm

Fig. 3 shows the BIDE algorithm. It first scans the database once to find the frequent 1-sequences (line 2), builds pseudo projected database for each frequent 1-sequence (line 3 and 4), treats each frequent 1-sequence as a prefix and uses *BackScan* pruning method to check if it can be pruned (line 6), if not, computes the number of *backward-extension-items* (line 7), and calls subroutine $bide(S_p-SDB, S_p, min_sup, BEI, FCS)$ (line 8). Subroutine $bide(S_p-SDB, S_p, min_sup, BEI, FCS)$ recursively calls itself and works as follows: For prefix S_p , scan its projected database S_p-SDB once to find its locally frequent items (line 10), compute the number of *forward-extension-items* (line 11), if there is no *backward-extension-item* nor *forward-extension-item*, output S_p as a frequent closed sequence (line 12 and 13), grow S_p with each locally frequent item in lexicographical ordering to get a new prefix (line 15) and build the pseudo projected database for the new prefix (line 16), for each new prefix, first check if it can be

pruned (line 18), if not, compute the number of *backward-extension-items* (line 19) and call itself (line 20). We need to point out that both subroutines *BackScan()* and *backward extension check()* use the *ScanSkip* technique to speed up the mining process.

```

BIDE ( $SDB, min\_sup, FCS$ )
Input: an input sequence database  $SDB$ , a minimum support threshold  $min\_sup$ 
Output: the complete set of frequent closed sequences,  $FCS$ 
1:  $FCS = \emptyset$ ;
2:  $FI$ =frequent 1-sequences( $SDB, min\_sup$ );
3: for (each 1-sequence  $fl$  in  $FI$ ) do
4:    $SDB^{fl}$ = pseudo projected database ( $SDB$ );
5:   for (each  $fl$  in  $FI$ ) do
6:     if ( $\neg BackScan(fl, SDB^{fl})$ )
7:        $BEI$ =backward extension check ( $fl, SDB^{fl}$ );
8:       call bide( $SDB^{fl}, fl, min\_sup, BEI, FCS$ );
9:   return  $FCS$ ;

bide ( $S_p, SDB, S_p, min\_sup, BEI, FCS$ )
Input: a projected sequence database  $S_p, SDB$ , a prefix sequence  $S_p$ ,
a minimum support threshold  $min\_sup$ , and the number of backward
extension items  $BEI$ 
Output: the current set of frequent closed sequences,  $FCS$ 
10:  $LFI$  = locally frequent items ( $S_p, SDB$ );
11:  $FEI = \{i \text{ in } LFI \mid z.\text{sup} = \text{sup}^{SDB}(S_p, i)\}$ ;
12: if ( $(BEI+FEI)=0$ )
13:    $FCS = FCS \cup \{S_p\}$ ;
14: for (each  $i$  in  $LFI$ ) do
15:    $S_p^i = \langle S_p, i \rangle$ ;
16:    $SDB^{S_p^i}$  = pseudo projected database ( $S_p, SDB, S_p^i$ );
17:   for (each  $i$  in  $LFI$ ) do
18:     if ( $\neg BackScan(S_p^i, SDB^{S_p^i})$ )
19:        $BEI$ =backward extension check ( $S_p^i, SDB^{S_p^i}$ );
20:       call bide( $SDB^{S_p^i}, S_p^i, min\_sup, BEI, FCS$ );

```

Fig. 3. BIDE algorithm.

3.6 Further discussions

Unlike most of the previous closed pattern mining algorithms, BIDE can prune search space and check if a pattern is closed without maintenance of any already mined patterns (or just candidates), it is very space efficient and its worst case memory usage can be calculated even before the mining. Assume the input sequence database SDB contains s_num input sequences (i.e., $|SDB|=s_num$) and totally i_num distinct items, on average each sequence contains avg_l events, the longest sequence contains max_l events. The worst case memory usage in BIDE is $O((s_num * avg_l) + (max_l * s_num * (2 * max_l)) + (max_l * i_num))$. ($s_num * avg_l$) represents the input sequence database. ($max_l * s_num * (2 * max_l)$) repre-

sents the worst case of space usage for building pseudo projected databases: the longest frequent closed sequence has a length no greater than max_l , which means there are at most max_l pseudo projected databases which can co-exist at a certain time, and each pseudo projected database corresponds to a prefix with a length no greater than max_l and a support no greater than s_num , given a prefix in any input sequence, we need to record at most $(2 * max_l)$ positions in order to locate all the last-in-last maximum periods (In our implementation, we only need to record $(max_l + 1)$ positions). ($max_l * i_num$) represents the upper bound space cost in computing the *backward-extension events*.

The above BIDE algorithm can only mine frequent closed sequences of single items, but it is rather easy to extend it to mine frequent closed sequences of subsets of items, that is, each event may contain a set of un-ordered items. As [4] has shown there are two kinds of extensions to grow a certain prefix sequence: sequence-extensions (abbreviated S-extension) and itemset-extensions (abbreviated I-extension). A sequence extension w.r.t. a prefix is generated by adding a new event consisting of a single item to the prefix sequence, while an itemset extension w.r.t. a prefix is a sequence generated by adding an item to any one event of the prefix sequence. To revise the frequent sequence enumeration algorithm shown in Fig. 2 in order to mine frequent sequences of subsets of items is straightforward, which is very similar to the pseudo-projection based PrefixSpan algorithm [17]. To check if a frequent sequence of subsets of items is closed, we need to figure out whether there exists any S-extension item or I-extension item which has the same support as the prefix sequence. The *BI-Directional Extension* closure checking scheme described in section 3.2 and the *BackScan* pruning method have shown how to compute the backward(or forward) S-extension items from the maximum periods (or semi-maximum periods), while it is a relatively straightforward process to extend it to compute the backward (or forward) I-extension items under the same framework. Due to limited space, we will leave it to the interested readers.

4 Performance evaluation

In this section, we will present our thorough experimental results in order to testify the following claims: (1) A properly designed frequent closed sequence mining algorithm like BIDE or CloSpan can significantly outperform two efficient frequent sequence mining algorithms, PrefixSpan and SPADE when the support threshold is low; (2) BIDE consumes much less memory and can be an order of magnitude faster than CloSpan; (3) BIDE has linear scalability in terms of base size; and (4) The *BackScan* pruning method and the *ScanSkip* technique are very effective in enhancing the performance.

4.1 Test environment and datasets

All of our experiments were performed on an IBM ThinkPad R31 with 384MB memory and Windows XP professional installed. In the experiments we compared BIDE with two frequent sequence mining algorithms, PrefixSpan and SPADE, and one frequent closed sequence mining algorithm, CloSpan. The source codes of SPADE and PrefixSpan and the executable code of CloSpan were provided by their corresponding authors, respectively. We ran the four algorithms on the same Cygwin environment with the output turned off.

Because the synthetic datasets have far different characteristics from the real-world ones, in our experiments we only used some real datasets to do the tests. However, we cautiously chose these real datasets which can cover a range of distribution characteristics: sparse, a little dense, and very dense.

The first dataset, *Gazelle*, is very sparse, but it contains some very long frequent closed sequences with low support threshold. This dataset was originally provided by Blue Martini company and has been used in evaluating both frequent itemset mining algorithms [25, 8] and frequent sequence mining algorithms [22]. It contains totally 29369 customers' Web click-stream data. For each customer there is a corresponding series of page views, and we treat each page view as an event. This dataset contains 29369 sequences (i.e., customers), 87546 events (i.e., page views), and 1423 distinct items (i.e., web pages). More detailed information about this dataset can be found in [11].

The second dataset, *Snake*, is a little dense and can generate a lot of frequent closed sequences with a medium support threshold like 60%. It is a bio-dataset which contains totally 175 Toxin-Snake protein sequences and 20 unique items. This Toxin-Snake dataset is about a family of eukaryotic and viral DNA binding proteins and has been used in evaluating pattern discovery task [10].

We also find a very dense dataset, *Pi*, from which a huge number of frequent closed sequences can be mined even with a very high support threshold like 90%. This dataset is also a bio-dataset which contains 190 protein sequences and 21 distinct items. This dataset has ever been used to assess the reliability of functional inheritance [3]. The characteristics of these datasets are shown in Table 2.

Dataset	# seq.	# items	avg. seq. len.	max. seq. len.
<i>Gazelle</i>	29369	1423	3	651
<i>Snake</i>	175	20	67	121
<i>Pi</i>	190	21	258	757

Table 2. Dataset Characteristics.

4.2 Experimental results

Closed vs. all frequent sequence mining. Previous study [22] has shown that a properly designed closed sequential pattern mining algorithm like CloSpan can outperform an efficient sequential pattern mining algorithm, PrefixSpan, by more than an order of magnitude. As a result, it will be unfair to compare BIDE with some All-frequent-sequence mining algorithms. Here we only use one dataset to show a previously un-revealed fact: with a high support threshold, a well-designed closed sequence mining algorithm may lose to some all-frequent-sequence mining algorithms. Later we will focus on the comparison of BIDE with CloSpan.

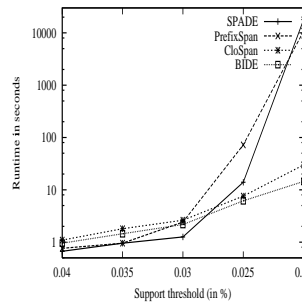


Fig. 4. Comparison among SPADE, PrefixSpan, CloSpan and BIDE (*Gazelle* dataset, runtime).

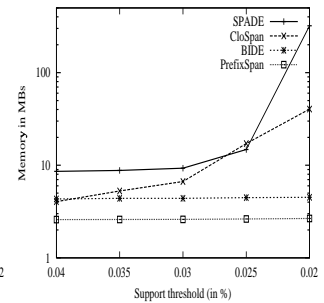


Fig. 5. Comparison among BIDE, CloSpan, SPADE and PrefixSpan (*Gazelle* dataset, memory).

Fig. 4 and Fig. 5 depict the comparison results among SPADE, PrefixSpan, CloSpan and BIDE for dataset *Gazelle*. From Fig. 4 we can see that when the support is greater than 0.03%, both PrefixSpan and Spade outperform CloSpan and BIDE, but once we continue to lower the support threshold (e.g., to 0.02%), the two closed sequence mining algorithms will outperform a lot the two all-frequent-sequence mining algorithms due to the generation of an explosive number of frequent sequences for the latter. It also shows that BIDE always outperforms CloSpan with varying support threshold. From Fig. 5 we know that both PrefixSpan and BIDE use a rather stable sized memory, which can be more than an order of magnitude less than those used by SPADE and CloSpan.

BIDE vs. CloSpan. We first compared BIDE with CloSpan using the *Gazelle* dataset. Fig. 6 depicts the distribution of the number of frequent closed sequences against the length of the frequent closed sequences for support thresholds varying from 0.02% to 0.01%. From Fig. 6 we can see that many long closed sequences can be discovered for this sparse dataset. For example, at support 0.01%, the longest frequent closed sequence has a length 127. Fig. 7 and Fig. 8 demonstrate the runtime and memory usage comparison

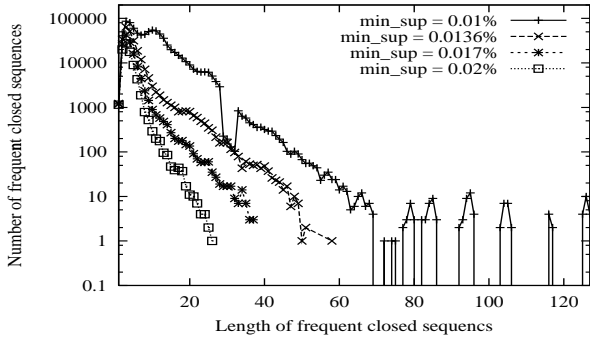


Fig. 6. Dataset *Gazelle*(distribution).

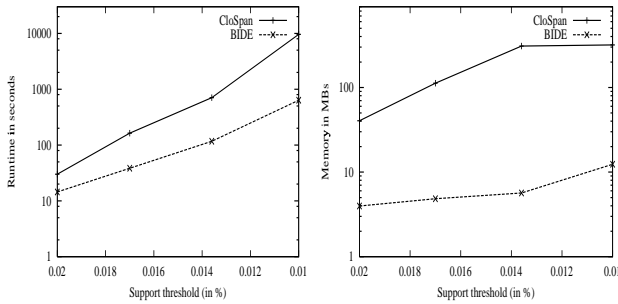


Fig. 7. Dataset *Gazelle*(runtime).

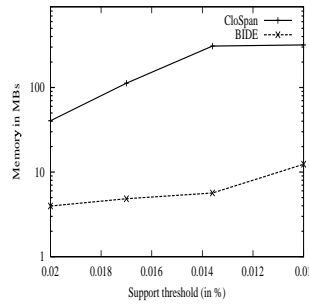


Fig. 8. Dataset *Gazelle*(memory).

between BIDE and CloSpan. We can see BIDE always runs much faster than CloSpan but consumes much less memory. For example, at support 0.01%, BIDE can be more than an order of magnitude faster than CloSpan, while it only uses over an order of magnitude less memory.

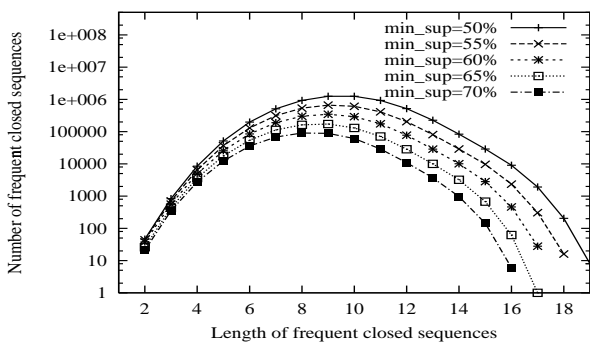


Fig. 9. Dataset *Snake*(distribution).

Fig. 9 depicts the distribution of the number of frequent closed sequences against the length of the frequent closed sequences for dataset *Snake*. We can see it is a little dense dataset: A lot of closed sequences with a medium length from 6 to 12 can be mined with some not very low sup-

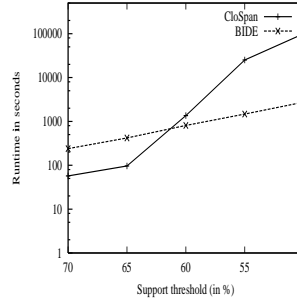


Fig. 10. Dataset *Snake*(runtime).

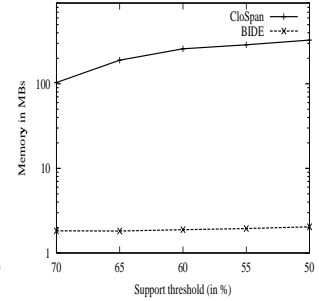


Fig. 11. Dataset *Snake*(memory).

port thresholds from 50% to 70%. Fig. 10 shows that at a high support threshold, BIDE is several times slower than CloSpan, but once the support is no greater than 60%, BIDE will significantly outperform CloSpan. For example, at support 50%, BIDE is about 40 times faster than CloSpan. From Fig. 11 we can see BIDE uses more than 2 orders of magnitude less memory than CloSpan in almost all the cases. Although this dataset only contains 175 sequences, which is rather small, however, because CloSpan needs to keep track of the already mined frequent closed sequence candidates, it can consume more than 300MB memory. For example, at support 50%, BIDE only uses about 2MB memory, while CloSpan uses about 328MB memory.

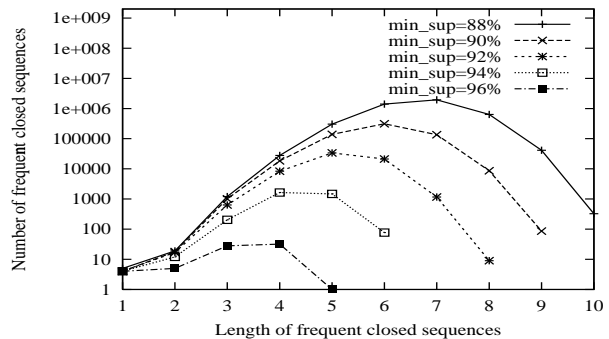


Fig. 12. Dataset *Pi*(distribution).

We also use the very dense dataset, *Pi*, to compare BIDE with CloSpan. From Fig. 12 we can see that even with a very high support like 90%, there can be a large number of short frequent closed sequences with a length less than 10. Fig. 13 shows that with a support higher than 90%, these two algorithms have very similar performance, CloSpan is only a little faster than BIDE, but once the support is no greater than 88%, BIDE will outperform CloSpan a lot. For example, at support 88%, BIDE is more than 6 times faster than CloSpan. From Fig. 14 we know BIDE always uses much less memory than CloSpan. At support 88%, BIDE

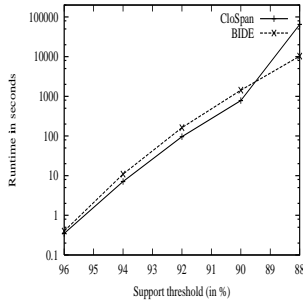


Fig. 13. Dataset *Pi* (runtime).

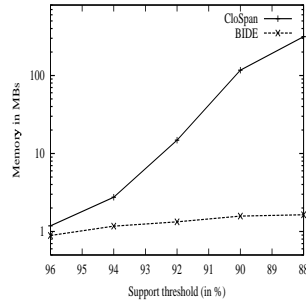


Fig. 14. Dataset *Pi* (memory).

consumes over 2 orders of magnitude less memory than CloSpan.

Scalability test. We tested BIDE’s scalability in both runtime and memory usage using all the three datasets in terms of the base size. In Fig. 15 and Fig. 16, we fixed the support threshold at a certain constant for each dataset and replicated the sequences from 2 to 16 times. Although these 3 datasets have rather different features, BIDE shows a linear scalability in both the runtime and memory usage against the increasing number of sequences for these datasets. For example, for dataset *Snake* with a given support 60%, its runtime increases from 807 seconds to 11906 seconds and its memory usage increases from 1.883MB to 21.784MB when the number of sequences increases 16 times.

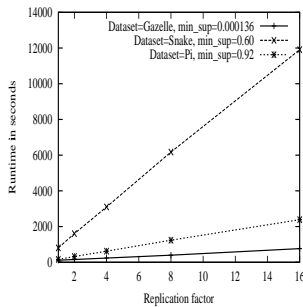


Fig. 15. Scalability test(runtime).

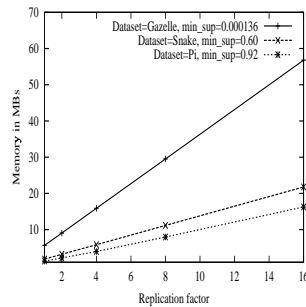


Fig. 16. Scalability test(memory).

Effectiveness of the optimization techniques. Fig. 17 tests the effectiveness of the *BackScan* pruning method. We can see that the *BackScan* pruning method is very effective in pruning search space and speeding up the mining process: for *gazelle* dataset with support threshold at 0.02%, it can give several orders of magnitude enhancement to the performance. The effectiveness of the *BackScan* pruning method assures that BIDE which is only based on this single pruning method can significantly outperform in most cases the CloSpan algorithm which adopts several pruning methods.

Fig. 18 shows the effectiveness of the *ScanSkip* optimization technique for *Snake* dataset. This technique always makes BIDE run faster with varying support thresholds. All the above performance results for BIDE use both the *BackScan* pruning method and the *ScanSkip* technique.

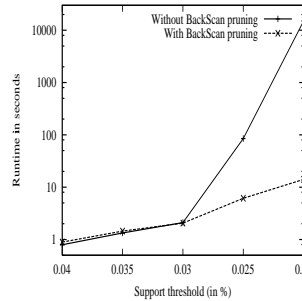


Fig. 17. Effectiveness of *BackScan* pruning (*Gazelle* dataset).

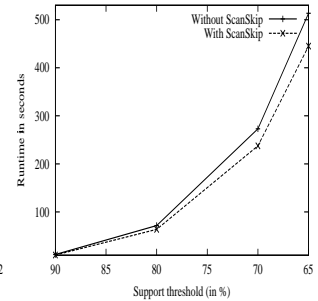


Fig. 18. Effectiveness of *ScanSkip* optimization (*Snake* dataset).

5 Conclusions

Many studies have elaborated that closed pattern mining has the same expressive power as that of all frequent pattern mining yet leads to more compact result set and significantly better efficiency. Our study showed that this is usually true when the number of frequent patterns is prohibitively huge, in which case the number of frequent closed patterns is also likely very large. Unfortunately, most of the previously developed closed pattern mining algorithms rely on the historical set of frequent closed patterns (or candidates) to check if a newly found frequent pattern is closed or if it can invalidate some already mined closed candidates. Because the set of already mined frequent closed patterns keeps growing during the mining process, not only will it consume more memory, but also lead to inefficiency due to the growing search space for pattern closure checking.

In this paper, we proposed BIDE, a novel algorithm for mining frequent closed sequences. It avoids the curse of the *candidate maintenance-and-test* paradigm, prunes the search space more deeply and checks the pattern closure in a more efficient way while consuming much less memory in contrast to the previously developed closed pattern mining algorithms. It does not need to maintain the set of historic closed patterns, thus it scales very well in the number of frequent closed patterns. BIDE adopts a strict depth-first search order and can output the frequent closed patterns in an online fashion. An extensive set of experiments on several real datasets with different distribution features have shown the effectiveness of the algorithm design: BIDE consumes order(s) of magnitude less memory while can be

over an order of magnitude faster than the CloSpan algorithm. It also has linear scalability in terms of the number of sequences in the database. Many studies have shown that constraints are essential for many sequential pattern mining applications. In the future, we plan to study how to push constraints (like gap constraint) into BIDE in order to mine more compact and specific result set.

Acknowledgment

We are grateful to Dr. Mohammed Zaki for providing us the source code of SPADE. Thanks also go to Dr. George Karypis for providing us the source of some protein sequence data, and Xifeng Yan for providing us the executable code of CloSpan and some helpful discussions.

References

- [1] R. Agrawal and R. Srikant, *Fast algorithms for mining association rules*. In VLDB'94, Santiago, Chile, Sept. 1994.
- [2] R. Agrawal, and R. Srikant, *Mining sequential patterns*. In ICDE'95, Taipei, Taiwan, Mar. 1995.
- [3] P. Aloy, E. Querol, F.X. Aviles and M.J.E. Sternberg, *Automated Structure-based Prediction of Functional Sites in Proteins: Applications to Assessing the Validity of Inheriting Protein Function From Homology in Genome Annotation and to Protein Docking*. Journal of Molecular Biology, 311, 2002.
- [4] J. Ayres, J. Gehrke, T. Yiu, and J. Flannick, *Sequential Pattern Mining using a Bitmap Representation*. In SIGKDD'02, Edmonton, Canada, July 2002.
- [5] C. Bettini, X. Wang, and S. Jajodia, *Mining temporal relationals with multiple granularities in time sequences*. Data Engineering Bulletin, 21(1):32-38, 1998.
- [6] M. Garofalakis, R. Rastogi, and K. Shim, *SPIRIT: Sequential Pattern Mining with regular expression constraints*. In VLDB'99, San Francisco, CA, Sept. 1999.
- [7] J. Han, G. Dong, and Y. Yin, *Efficient mining of partial periodic patterns in time series database*. In ICDE'99, Sydney, Australia, Mar. 1999.
- [8] J. Han, J. Wang, Y. Lu, and P. Tzvetkov, *Mining Top-K Frequent Closed Patterns without Minimum Support*. In ICDM'02, Maebashi, Japan, Dec. 2002.
- [9] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.C. Hsu, *FreeSpan: Frequent pattern-projected sequential pattern mining*. In SIGKDD'00, Boston, MA, Aug. 2000.
- [10] I. Jonassen, J.F. Collins, and D.G. Higgins, *Finding flexible patterns in unaligned protein sequences*. Protein Science, 4(8), 1995.
- [11] R. Kohavi, C. Brodley, B. Frasca, L.Mason, and Z. Zheng, *KDD-cup 2000 organizers' report: Peeling the Onion*. SIGKDD Explorations, 2, 2000.
- [12] H. Mannila, H. Toivonen, and A.I. Verkamo, *Discovering frequent episodes in sequences*. In SIGKDD'95, Montreal, Canada, Aug. 1995.
- [13] F. Massegli, F. Cathala, and P. Poncelet, *The psp approach for mining sequential patterns*. In PKDD'98, Nantes, France, Sept. 1995.
- [14] B. Ozden, S. Ramaswamy, and A. Silberschatz, *Cyclic association rules*. In ICDE'98, Orlando, FL, Feb. 1998.
- [15] N. Pasquier, Y. Bastide, R. Taouil and L. Lakhal, *Discovering frequent closed itemsets for association rules*. In ICDT'99, Jerusalem, Israel, Jan. 1999.
- [16] J. Pei, J. Han, and R. Mao, *CLOSET: An efficient algorithm for mining frequent closed itemsets*. In DMKD'01 workshop, Dallas, TX, May 2001.
- [17] J. Pei, J. Han, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.C. Hsu, *PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth*. In ICDE'01, Heidelberg, Germany, April 2001.
- [18] J. Pei, J. Han, and W. Wang, *Constraint-based sequential pattern mining in large databases*. In CIKM'02, McLean, VA, Nov. 2002.
- [19] M. Seno, G. Karypis, *SLPMiner: An algorithm for finding frequent sequential patterns using length-decreasing support constraint*. In ICDM'02, Maebashi, Japan, Dec. 2002.
- [20] R. Srikant, and R. Agrawal, *Mining sequential patterns: Generalizations and performance improvements*. In EDBT'96, Avignon, France, Mar. 1996.
- [21] J. Wang, J. Han, and J. Pei, *CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets*. In KDD'03, Washington, DC, Aug. 2003.
- [22] X. Yan, J. Han, and R. Afshar, *CloSpan: Mining Closed Sequential Patterns in Large Databases*. In SDM'03, San Francisco, CA, May 2003.
- [23] J. Yang, P.S. Yu, W. Wang and J. Han, *Mining long sequential patterns in a noisy environment*. In SIGMOD'02, Madison, WI, June 2002.
- [24] M. Zaki, *SPADE: An Efficient Algorithm for Mining Frequent Sequences*. Machine Learning, 42:31-60, Kluwer Academic Publishers, 2001.
- [25] M. Zaki, and C. Hsiao, *CHARM: An efficient algorithm for closed itemset mining*. In SDM'02, Arlington, VA, April 2002.