Frequent Closed Sequence Mining without Candidate Maintenance

Jianyong Wang, *Senior Member*, *IEEE*, Jiawei Han, *Senior Member*, *IEEE*, and Chun Li

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 a 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. A thorough performance study with both sparse and dense, real, and synthetic data sets has demonstrated that BIDE significantly outperforms the previous algorithm: It consumes an 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.

Index Terms—Data mining, frequent closed sequences, BI-Directional Extension.

1 Introduction

Sequential pattern mining, since its introduction in [2], has become an essential data mining task with broad applications, including feature selection for sequence classification [8], [26], mining minimal distinguishing subsequence patterns [12], mining block correlations in storage systems [15], finding copy-paste and related bugs in large-scale software code [16], and mining API usages from open source repositories [30], [31]. Efficient mining methods have been studied extensively, including the general sequential pattern mining [18], [27], [11], [34], [22], [4], constraint-based sequential pattern mining [9], [23], [25], top-k closed sequential pattern mining [28], parallel closed sequential pattern mining [7], frequent episode mining [17], cyclic association rule mining [19], long sequential pattern mining in noisy environment [33], and frequent partial order mining [6], [24].

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 a more compact yet complete result set but also better efficiency [20], [35], [32], [29]. 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 the best of our knowledge, CloSpan is currently the only such algorithm [32]. Like most of the frequent closed itemset mining algorithms, it follows a *candidate maintenance-and-test* paradigm, that is, it needs to maintain the set of already mined closed sequence candidates that can be used to prune the 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 that leads to an algorithm, BIDE,[1] that efficiently mines 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 techniques. Our thorough performance study demonstrates the big success of the algorithm design: BIDE consumes an 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 and the

- *J. Wang and C. Li are with the Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China. E-mail: jianyong@tsinghua.edu.cn, Socrates.lee@gmail.com.*
- *J. Han is with the Department of Computer Science, University of Illinois at Urbana-Champaign, 201 N. Goodwin Avenue, 2132 Siebel Center for Computer Science, MC-258, Urbana, IL 61801-2302. E-mail: hanj@cs.uiuc.edu.*

Manuscript received 18 Sept. 2006; revised 5 Mar. 2007; accepted 30 Mar. 2007; published online 3 Apr. 2007.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0435-0906.
Digital Object Identifier no. 10.1109/TKDE.2007.1043.

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

TABLE 1
An Example Sequence Database SDB

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

BackScan pruning method. In this section, we also discuss how to extend BIDE to mine frequent closed sequences of subsets of items from general sequence databases. In Section 4, we present an extensive experimental study. Finally, we conclude the study in Section 5.

2 Problem Definition and Related Work

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of distinct items. A **sequence** S is an ordered list of events, denoted by $\langle e_1, e_2, \ldots, e_m\rangle$, where e_i is an item, that is, $e_i \in I$ for $1 \leq i \leq m$. For brevity, a sequence is also written as $e_1e_2 \ldots e_m$. From the definition, we know that an item can occur multiple times in different events of a sequence. The number of events (that is, 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 $S_a = a_1a_2 \ldots a_n$ is *contained* in another sequence $S_b = b_1b_2 \ldots b_m$ if there exist integers $1 \leq i_1 < i_2 < \ldots < i_n \leq m$ such that $a_1 = b_{i_1}, a_2 = b_{i_2}, \ldots, a_n = b_{i_n}$. 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 by $S_a \sqsubseteq S_b$.

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 by $|SDB|$. A tuple (sid, S) is said to *contain* a sequence S_α if S is a supersequence of S_α, that is, $S_\alpha \sqsubseteq S$. 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 by $sup^{SDB}(S_\alpha)$, and the **relative support** is the percentage of tuples in SDB that *contain* S_α (that is, $sup^{SDB}(S_\alpha)/|SDB|$). 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 remainder 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, that is, there exists no S_β 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 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 a total of three unique items and four input sequences (that is, $|SDB| = 4$). Suppose $min\_sup = 2$. The complete set of frequent closed sequences $S_{fcs} = \{\text{AA:2, ABB:2, ABC:4, CA:3, CABC:2, CB:3}\}$ consists of only six sequences, whereas the whole set of frequent sequences consists of 17 sequences, that is,

$$S_{fs} = \{\text{A:4, AA:2, AB:4, ABB:2, ABC:4, AC:4, B:4, BB:2, BC:4, C:4, CA:3, CAB:2, CABC:2, CAC:2, CB:3, CBC:2, CC:2}\}.$$

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 supersequences 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 3.5, we discuss how to extend BIDE to mine closed sequences of subsets of items (for example, sequences of shopping transactions). The reasons why we first discuss mining single-item sequences include the following: 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, Generalized Sequential Patterns (GSP) [27], based on the Apriori property [1]. Since then, many sequential pattern mining algorithms have also been proposed for performance improvements. Among those, SPADE [34], PrefixSpan [22], 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, whereas PrefixSpan adopts a horizontal format data set 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 developed 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 [20], many efficient frequent closed itemset mining algorithms have been proposed, such as A-Close [20], CLOSET [21], CHARM [35], CLOSET+ [29], and several recently devised methods [5]. 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 [10] and CLOSET+,[2] adopt a compact two-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 [32]. It follows the *candidate maintenance-and-test* approach: First, generate a set of closed sequence candidates that is stored in a hash-indexed result-tree structure and then do postpruning on it. It uses some pruning methods like *CommomPrefix* 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 a 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 the following: 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, whereas the backward directional extension can be used to both check the closure of a prefix pattern and prune the search space. 2) Under the BI-Directional Extension paradigm, the *BI-Directional Extension* pattern closure checking scheme and the *BackScan* pruning method are proposed to speed up the mining and also assure the correctness of the mining algorithm. We also devise the BIDE algorithm in order to mine frequent closed sequences.

3 BIDE: An Efficient Algorithm for Frequent Closed Sequence Mining

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

3.1 Frequent Sequence Enumeration

Assume that there is a lexicographical ordering $\leq$ among the set of items I in the input sequence database (for example, 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 nonclosed ones.

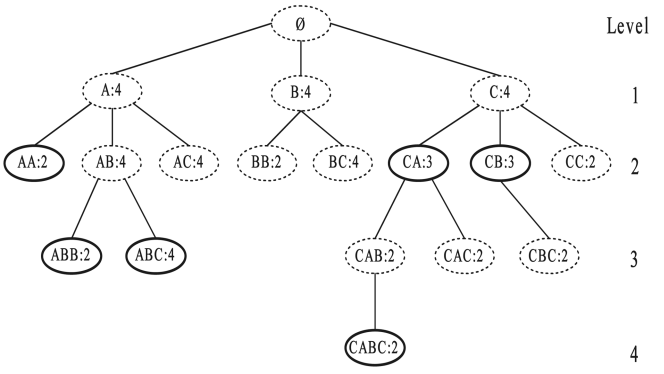

Fig. 1. The lexicographic frequent sequence tree in our running example.

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: $A{:}4$, $AA{:}2$, $AB{:}4$, $ABB{:}2$, $ABC{:}4$, $AC{:}4$, $B{:}4$, $BB{:}2$, $BC{:}4$, $C{:}4$, $CA{:}3$, $CAB{:}2$, $CABC{:}2$, $CAC{:}2$, $CB{:}3$, $CBC{:}2$, $CC{:}2$.

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 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 with respect to 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 *pseudoprojection* [22]. 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 pseudoprojection method to find the set of locally frequent items with respect to a certain prefix and use them to grow the corresponding prefix. Here, we briefly introduce the pseudoprojection method (for details, see [22]).

Definition 1 (First instance of a prefix sequence). *Given an input sequence S that contains a prefix 1-sequence e_1, the subsequence from the beginning of S to the first appearance of*

2. CLOSET+ adopts a hybrid closure checking scheme: The *result-tree* method for dense data sets and *upward checking* for sparse data sets, 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.

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 *Frequent-sequences(SDB, Ø, min_sup, FS)*;
3: return *FS*;

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 *Frequent-sequences*(S_p^i_*SDB*, S_p^i, *min_sup*, *FS*);

Fig. 2. Frequent sequence enumeration algorithm.

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 $e_1e_2 \dots e_ie_{i+1}$ 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} that 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 that 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 with respect to 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 with respect to a prefix sequence $e_1e_2 \dots e_i$ is called the projected database with respect to 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 pseudoprojection 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. Moreover, by scanning forward each projected sequence with respect to a prefix S_p and counting the items (this is the so-called *Forward-extension* step), we will find the locally frequent items with respect to 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 $\{C{:}4,\ B{:}2\}$.

Fig. 2 shows the algorithm to enumerate the complete set of frequent sequences, which is similar to the pseudoprojection-based PrefixSpan algorithm. It recursively calls subroutine *Frequent-sequences* (S_p_SDB, S_p, min_sup, and FS): For a certain prefix S_p, if it is nonempty, output it (lines 4 and 5), scan projected database S_p_SDB once to find the locally frequent items (line 6), choose each frequent item i in lexicographical ordering to grow S_p to get a new prefix S_p^i (line 10), and scan S_p_SDB once again to build the pseudoprojection 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, that is, it cannot be *absorbed* by one of its supersequence with the same support. Currently, most of the frequent closed pattern (both itemset and sequence) mining algorithms like CLOSET [21], CHARM [35], TFP [10], and CloSpan [32] need to maintain the set of already mined frequent closed patterns (or just candidates) in memory and do 1) *subpattern checking*, which checks if a newly found pattern can be absorbed by an already mined frequent closed pattern (or candidate), and/or 2) *superpattern 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 superpattern checking. However, a typical frequent closed sequence mining algorithm usually needs to do both subpattern and superpattern 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 superpattern checking in order to remove the previously mined but nonclosed 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 subpattern checking in order to remove the newly found but nonclosed 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, 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 [10] 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 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 nonclosed, there must exist at least one event e' that can be used to extend sequence S to get a new sequence S' that has the same support. The sequence S can be extended in three ways: 1) $S' = e_1e_2 \dots e_ne'$ and $sup^{SDB}(S') = sup^{SDB}(S)$, 2) $\exists i\ (1 \leq i < n)$, $S' = e_1e_2 \dots e_ie'e_{i+1} \dots e_n$, and $sup^{SDB}(S') = sup^{SDB}(S)$, and 3) $S' = e'e_1e_2 \dots e_n$ 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* with respect to S. Whereas 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* with respect to 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 with respect to a prefix sequence S_p, S_p must be a closed sequence; otherwise, S_p must be nonclosed.*

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 with respect to 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 with respect to S_p. □

Definition 4 (Last instance of a prefix sequence). *Given an input sequence S that 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 ith last-in-last appearance with respect to a prefix sequence). *For an input sequence S containing a prefix n-sequence $S_p = e_1e_2 \dots e_n$, the ith last-in-last appearance with respect to the prefix S_p in S is denoted by LL_i and defined recursively as follows: 1) If $i = n$, it is the last appearance of e_i in the last instance of the prefix S_p in S, and 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, and LL_i must appear before LL_{i+1}. For example, if $S = CAABC$ and $S_p = AB$, the first last-in-last appearance with respect to prefix S_p in S is the second A in S; whereas if $S = CACAC$ and $S_p = CAC$, the first last-in-last appearance with respect to prefix S_p in S is the second C in S, the second last-in-last appearance with respect to prefix S_p in S is the second A in S, and the third last-in-last appearance with respect to prefix S_p in S is the third C in S.*

Definition 6 (The ith 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 ith maximum period of the prefix S_p in S is defined as follows: 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 ith last-in-last appearance with respect to prefix S_p, and 2) if $i = 1$, it is the piece of sequence in S located before the first last-in-last appearance with respect to prefix S_p. For example, if $S = ABCB$ and $S_p = AB$, the first maximum period of prefix S_p in S is empty, and the second maximum period of prefix S_p in S is BC; whereas if $S = ABBB$ and $S_p = BB$, the first maximum period of prefix S_p in S is AB, and the second maximum period of prefix S_p in S is B.*

Lemma 2 (Backward-extension event checking). *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' that appears in each of the ith maximum periods of the prefix S_p in SDB, e' is a backward-extension event (or item) with respect to prefix S_p. Otherwise, for any $i\ (1 \leq i \leq n)$, if we cannot find any item that appears in each of the ith maximum periods of the prefix S_p in SDB, there will be no backward-extension event with respect to prefix S_p.*

Proof. From the definition of the ith maximum period of a prefix sequence, we know that if item e' appears in each of the ith 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 $S'_p \sqsubset S_p$ and $sup^{SDB}(S'_p) = sup^{SDB}(S_p)$. Therefore, e' is a backward-extension item with respect to prefix S_p, and S_p is not closed.

In addition, assume that 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 with respect to 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 ith last-in-last appearance with respect to S_p, which means item e' must appear in the ith maximum period of S_p. As a result, for any $i\ (1 \leq i \leq n)$, if we cannot find any item that appears in each of the ith maximum periods of the prefix S_p in SDB, there will be no backward-extension event with respect to prefix S_p.□

Suppose the current prefix $S_p = e_1e_2 \dots e_n$ is contained in k sequences of database SDB, and there exists a backward-extension item e that can be used to extend S_p and get a new prefix $S'_p = e_1e_2 \dots e_{i-1}ee_i \dots e_n$, then item e must appear after the first appearance of subsequence $e_1e_2 \dots e_{i-1}$ but before the last appearance of subsequence $e_i \dots e_n$ in each of the k sequences containing S_p. This is the basic intuition behind Lemma 2. We use examples to illustrate the sequence closure checking scheme in BIDE. First, we

assume that $S_p = AC{:}4$; it is easy to find that item B appears in each of the second maximum periods of S_p with respect to SDB. As a result, $AC{:}4$ is not closed. In contrast, let $S_p = ABC{:}4$; we cannot find any backward-extension item with respect to SDB 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 nonredundant frequent sequences. Although the closure checking scheme can lead to a 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 that 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 (that is, $CA{:}3$, $CABC{:}2$, 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 ith last-in-first appearance with respect to a prefix sequence). *For an input sequence S containing a prefix n-sequence $S_p = e_1e_2 \dots e_n$, the ith last-in-first appearance with respect to the prefix S_p in S is denoted by LF_i and defined recursively as follows: 1) If $i = n$, it is the last appearance of e_i in the first instance of the prefix S_p in S, and 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, and LF_i must appear before LF_{i+1}. For example, if $S = CAABC$ and $S_p = CA$, the first last-in-first appearance with respect to prefix S_p in S is the first C in S, and the second last-in-first appearance with respect to prefix S_p in S is the first A in S; whereas if $S = CAABB$ and $S_p = CAB$, the first last-in-first appearance with respect to prefix S_p in S is the first C in S, the second last-in-first appearance with respect to prefix S_p in S is the second A in S, and the third last-in-first appearance with respect to prefix S_p in S is the first B in S.*

Definition 8 (The ith semimaximum period of a prefix sequence). *For an input sequence S containing a prefix n-sequence $S_p = e_1e_2 \dots e_n$, the ith semimaximum period of the prefix S_p in S is defined as follows: 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 ith last-in-first appearance with respect to prefix S_p, and 2) if $i = 1$, it is the piece of sequence in S located before the first last-in-first appearance with respect to prefix S_p. For example, if $S = ABCB$ and the prefix sequence $S_p = AC$, the first semimaximum period of prefix AC in S is empty, and the second semimaximum period of prefix AC in S is B; whereas if $S = CAABB$ and $S_p = CAB$, the first semimaximum period of prefix CAB in S is empty, the second semimaximum period of prefix CAB in S is A, and the third semimaximum period of prefix CAB in S is A.*

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' that appears in each of the ith semimaximum 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 ith semimaximum 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'' with respect to prefix S_p is also a locally frequent item with respect to 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 that 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 that appears in each of the first semimaximum period of prefix S_p in SDB and, so, we can safely stop mining frequent closed sequences with prefix $B{:}4$. In contrast, if $S_p = C{:}4$, we cannot find any item that appears in each of the first semimaximum periods of prefix S_p in SDB. As a result, we cannot stop growing $C{:}4$. One may wonder why the **BackScan search-space pruning** is based on semimaximum periods instead of maximum periods. Here, we use a counterexample to illustrate why backward-extension items found over the maximum periods cannot be used to stop growing the current prefix. Suppose the input sequence database contains only two identical sequences $S_1 = CAABC$ and $S_2 = CAABC$, and the current prefix $S_p = CA$. According to Definition 8, we cannot find any backward-extension item either from the set of the first semimaximum periods or from the set of the second semimaximum periods. Thus, we cannot prune prefix CA according to Theorem 2. However, we do find a backward extension item A that appears in each of the second maximum periods; if we stop growing prefix CA accordingly, we will no longer be able to discover the closed sequence $CAABC{:}2$.

Compared with some pruning methods used in previously developed algorithms [21], [35], [32], 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

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: *F1*=frequent 1-sequences(*SDB, min_sup*);
3: for (each 1-sequence *f1* in *F1*) do
4: SDB^{f1}= pseudo projected database (*SDB*);
5: for (each *f1* in *F1*) do
6: if (!*BackScan*(*f1*, SDB^{f1}))
7: *BEI*=backward extension check (*f1*, SDB^{f1});
8: call *bide*(SDB^{f1}, *f1, 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* = $\left|\{z \text{ in } LFI \mid z.\text{sup} = \sup^{SDB}(S_p)\}\right|$;
12: if ((*BEI*+*FEI*)==0)
13: *FCS*=*FCS* $\cup$ {S_p};
14: for (each *i* in *LFI*) do
15: S_p^i = <S_p,*i*>;
16: SDB^{Spi}= pseudo projected database (S_p_*SDB*, S_p^i);
17: for (each *i* in *LFI*) do
18: if (!*BackScan*(S_p^i, SDB^{Spi}))
19: *BEI*=backward extension check (S_p^i, SDB^{Spi});
20: call *bide*(SDB^{Spi}, S_p^i, *min_sup, BEI, FCS*);

Fig. 3. BIDE algorithm.

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. Although Theorem 2 provides an effective way for pruning some futile parts of the search space, we must remark here that only serves as a sufficient condition but not a necessary condition for pruning. That is, even if we cannot generate any closed sequences by growing the current prefix, we may not be able to prune the prefix sequence according to Theorem 2. This means that there is still some room to further prune some unpromising parts of the search space. In the following, we give a simple example where Theorem 2 cannot handle well. Suppose the sequence database contains two sequences $S_1 = CACB$ and $S_2 = CCACB$, and the current prefix $S_p = CC$. It is evident that we cannot rely on Theorem 2 to prune prefix CC; however, no closed sequences can be grown from prefix CC, as all the sequences with prefix CC (that is, CC and CCB) are nonclosed.

3.4 The BIDE Algorithm

Fig. 3 shows the BIDE algorithm. It first scans the database once to find the frequent 1-sequences (line 2), builds a pseudoprojected database for each frequent 1-sequence (lines 3 and 4), treats each frequent 1-sequence as a prefix and uses the *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 (lines 12 and 13), grow S_p with each locally frequent item in lexicographical ordering to get a new prefix (line 15), and build the pseudoprojected 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).

3.5 Closed Sequence Mining for General Sequence Databases

The BIDE algorithm shown in Fig. 3 can only mine frequent closed sequences of single items, but, in some cases, we need to mine frequent closed sequences of subsets of items, that is, each event may contain a set of unordered items. As the work in [4] has shown, there are two kinds of extensions to grow a certain prefix sequence: sequence extensions (S-extension) and itemset extensions (I-extension). A **sequence extension** with respect to a prefix is generated by adding a new event consisting of a single item to the prefix sequence, whereas an **itemset extension** with respect to a prefix is a sequence generated by adding an item to any one event of the prefix sequence. Revising the frequent sequence enumeration algorithm shown in Fig. 2 to mine frequent sequences of subsets of items is straightforward, which is the same as the pseudoprojection-based PrefixSpan algorithm [22]. In the following, we will mainly focus on how to adapt the *BI-Directional Extension* closure checking scheme for frequent closed sequence mining in general sequence databases. As the *BackScan* pruning method is very similar to the backward-extension event/item checking of the *BI-Directional Extension* closure checking scheme, we will not introduce its new form for general sequence databases. Also, due to limited space, we do not elaborate on the proofs (or rationale) of the corresponding theorems and lemmas. By adopting the general form of *BI-Directional Extension* closure checking scheme and *BackScan* pruning method introduced in this section, it is straightforward to adapt the BIDE algorithm to mine frequent closed sequences of subsets of items.

3.5.1 Generalized Closure Checking Scheme

To check if the current prefix sequence of subsets of items is closed or not, we need to figure out whether there exists any S-extension item or I-extension item that has the same support as the prefix sequence. Thus, we need to revise Theorem 1 to the following form.

Theorem 3 (Generalized BI-Directional Extension closure checking). *If there exists no forward-S-extension item, forward-I-extension item, backward-S-extension item, nor backward-I-extension item with respect to a prefix sequence S_p, S_p must be a closed sequence; otherwise, S_p must be nonclosed.*

TABLE 2
A General Sequence Database (GSDB)

Sequence identifier	Sequence
1	(A B D) (B C D) A
2	B (A B E) B
3	(A B) (B C D)

Suppose $S_p = e_1e_2 \dots e_m$, where e_i $(\forall i, 1 \leq i \leq m)$ is a subset of items $(x_{i_1}x_{i_2} \dots x_{i_k})$. We assume that the items in each event are sorted in lexicographical ordering. In Theorem 3, a **forward-S-extension item** x with respect to prefix S_p is an item that can be used to extend S_p as an S-extension to get a new prefix S'_p such that $S'_p = e_1e_2 \dots e_mx$ and $SUP^{SDB}(S_p) = SUP^{SDB}(S'_p)$, a **forward-I-extension item** x with respect to S_p is an item that can be used to extend S_p as an I-extension to get a new prefix S'_p such that $S'_p = e_1e_2 \dots (e_mx)$ and $SUP^{SDB}(S_p) = SUP^{SDB}(S'_p)$, a **backward-S-extension item** x with respect to S_p is an item that can be used to extend S_p as an S-extension to get a new prefix S'_p such that $S'_p = e_1e_2 \dots e_{i-1}xe_i \dots e_m$ $(\forall i, 1 \leq i \leq m)$ and $SUP^{SDB}(S_p) = SUP^{SDB}(S'_p)$, and a **backward-I-extension item** x with respect to S_p is an item that can be used to extend S_p as an I-extension to get a new prefix S'_p such that

$$S'_p = e_1e_2 \dots (x_{i_1} \dots x_{i_{j-1}}\ x\ x_{i_j} \dots x_{i_k}) \dots e_m\ (\forall j, 1 \leq j \leq (k+1)$$

if $1 \leq i < m$, whereas $1 \leq j \leq k$ if $i = m$) and

$$SUP^{SDB}(S_p) = SUP^{SDB}(S'_p).$$

Similarly, Lemma 1 should be revised to the following form.

Lemma 3 (Generalized forward-extension checking). *For a prefix sequence S_p, its complete set of forward-S-extension items and forward-I-extension items is equivalent to the set of its locally frequent S-extension and I-extension items whose supports are equal to $SUP^{SDB}(S_p)$.*

According to Lemma 3, it is very easy to compute the complete set of forward-extension items with respect to the current prefix S_p, which is equivalent to the set of the locally frequent S-extension and I-extension items whose supports are equal to $SUP^{SDB}(S_p)$, and the method to compute the locally frequent S-extension and I-extension items can be found in [22]. In the GSDB database shown in Table 2, if $S_p = A$, item B is both a forward-S-extension item and a forward-I-extension item with respect to S_p, thus S_p is not closed. As another case, we let $S_p = AB$, it is relatively easy to get that there is neither forward-S-extension item nor forward-I-extension item with respect to prefix AB. To judge whether AB is closed or not, we need to determine if we can find any backward-S-extension item or backward-I-extension item.

To facilitate the checking of backward-S-extension items and backward-I-extension items, we need to first define the **projected event** and **event appearance** of an event e_1 with respect to another event e_2.

Definition 9 (Projected event and event appearance). *Given two events $e_1 = (x_{1_1} \dots x_{1_m})$ and $e_2 = (x_{2_1} \dots x_{2_n})$, if $e_1 \subseteq e_2$, the subset of items of $(e_2 - e_1)$ is called the **projected event** of event e_1 with respect to e_2; otherwise, the projected event of e_1 with respect to e_2 is empty. In addition, in case of $e_1 \subseteq e_2$, e_2 is called an **appearance** of e_1. For example, let $e_1 =$ (A C) and $e_2 =$ (A B C D), the projected event of e_1 with respect to e_2 is (B D) and e_2 is an appearance of e_1.*

Given the above definition of an event appearance, Definition 4, Definition 5, Definition 6, and Lemma 2 still hold, but they can only be used to compute the set of backward-S-extension items for a prefix sequence. To compute the set of backward-I-extension items, we need to devise a new method.

Definition 10 (The ith I-extension period of a prefix sequence). *For an input sequence S containing a prefix $S_p = e_1e_2 \dots e_n$, the ith I-extension period of the prefix S_p in S is defined as follows: 1) If $1 < i < n$, 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 beginning of the first event in S after the ith last-in-last appearance with respect to prefix S_p, 2) if $i = 1$, it is the piece of sequence in S located before the first event in S after the first last-in-last appearance with respect to prefix S_p, and 3) if $i = n$, it is the piece of sequence in S after the first instance of prefix $e_1e_2 \dots e_{n-1}$ but before the first event in S after the nth last-in-last appearance with respect to prefix S_p.*

Definition 11 (The ith backward-I-extension itemset of a prefix sequence). *For an input sequence S containing a prefix $S_p = e_1e_2 \dots e_n$, the union of the projected events of event e_i with respect to any event appearance of e_i in the ith I-extension period is called the ith backward-I-extension itemset of a prefix sequence S_p with respect to input sequence S.*

Note that in Definition 11, if $i = n$, we need to exclude from the nth I-extension itemset of prefix S_p the items that are lexicographically greater than any item in the nth event of S_p, as they have been already considered in the forward-I-extension item checking.

Lemma 4 (Backward-I-extension item checking). *Let the prefix sequence be $S_p = e_1e_2 \dots e_n$. If $\exists i$ $(1 \leq i \leq n)$ and there exists an item x that appears in each of the ith I-extension itemsets of the prefix S_p in SDB, x is a backward-I-extension item with respect to prefix S_p. Otherwise, for any i $(1 \leq i \leq n)$, if we cannot find any item that appears in each of the ith I-extension itemsets of the prefix S_p in SDB, there will be no backward-I-extension item with respect to prefix S_p.*

Example 2. We use the GSDB database shown in Table 2 as our running example and let $S_p =$ B:3. The first I-extension itemsets of prefix S_p with respect to input sequences 1, 2, and 3 are $\{\text{A D}\} \cup \{\text{C D}\} = \{\text{A C D}\}$, $\{\text{A E}\}$, and $\{\text{A}\} \cup \{\text{C D}\} = \{\text{A C D}\}$, respectively. Note that as B is also the last event of prefix S_p and items C, D, and E are lexicographically greater than item B, they should be excluded from the corresponding I-extension itemsets. The first I-extension itemsets of prefix S_p with respect to input sequences 1, 2, and 3 become $\{\text{A}\}$, $\{\text{A}\}$, and $\{\text{A}\}$. We see that item A is a backward-I-extension item with respect to prefix B:3 and database GSDB; thus, B:3 is not closed.

TABLE 3
Real Data Set Characteristics

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

4 Performance Evaluation

In this section, we will present our thorough experimental results in order to verify the following claims: 1) BIDE consumes much less memory and can be an order of magnitude faster than CloSpan when the support is low, 2) BIDE has linear scalability against base size in terms of runtime efficiency and space usage, and 3) the *BackScan* pruning method is very effective in enhancing the performance of BIDE.

4.1 Test Environment and Data Sets

All of our experiments except those regarding general frequent closed sequence mining were performed on an IBM ThinkPad R31 with a 384-Mbyte memory and Windows XP professional installed. In the experiments, we compared BIDE with the well-known closed sequential pattern mining algorithm CloSpan using both real and synthetic data sets.

The first data set, *Gazelle*, is very sparse, but it contains some very long frequent closed sequences with low support thresholds. This data set was originally provided by Blue Martini and has been used in evaluating both frequent itemset mining algorithms [35], [10] and frequent sequence mining algorithms [32]. It contains a total of 29,369 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 data set contains 29,369 sequences (that is, customers), 87,546 events (that is, page views), and 1,423 distinct items (that is, Web pages). More detailed information about this data set can be found in [14].

The second data set, *Snake*, is a little dense and can generate a lot of frequent closed sequences with a medium support threshold like 60 percent. It is a bio-data set, which contains a total of 175 Toxin-Snake protein sequences and 20 unique items. This Toxin-Snake data set is about a family of eukaryotic and viral DNA binding proteins and has been used in evaluating pattern discovery task [13].

The third data set, *Pi*, is very dense and can generate a huge number of frequent closed sequences even with a very high support threshold like 90 percent. This data set is also a bio-data set, which contains 190 protein sequences and 21 distinct items. This data set has been used to assess the reliability of functional inheritance [3]. The characteristics of these real data sets are shown in Table 3.

The fourth data set, C10T8S8I8, is a general sequence data set (of a set of sequences of subsets of items), which was generated using the IBM synthetic data generator. This data set has been adopted in some previous studies such as that in [22]. It contains 10,000 customers (that is, sequences), and the number of distinct items is 1,000. Both the average number of items in a transaction (that is, event) and the average number of transactions in a sequence are set to eight. On the average, a frequent sequential pattern consists of four transactions, and each transaction is composed of eight items.

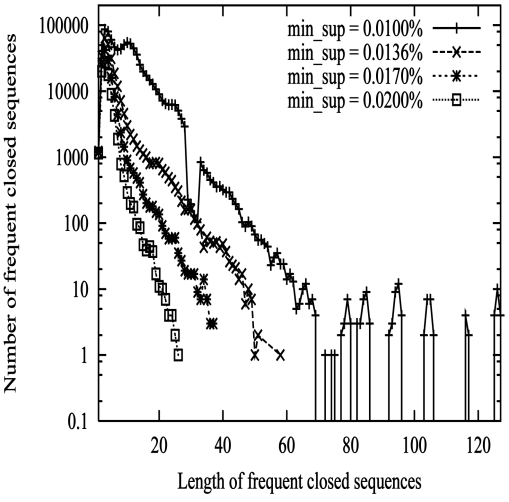

Fig. 4. Data set *Gazelle* (distribution).

To test the scalability of BIDE for mining frequent closed sequences of subsets of items, we generated a series of synthetic data sets using the IBM data generator, C200T2.5S10I1.25, C400T2.5S10I1.25, C600T2.5S10I1.25, C800T2.5S10I1.25, and C1000T2.5S10I1.25, which have a base size of 200,000, 400,000, 600,000, 800,000, and 1,000,000 sequences, respectively. All these data sets have the same distribution: the number of distinct items is 10,000, the average number of transactions (that is, events) in a sequence is 10, and the average number of items in a transaction is 2.5. On average, a frequent sequential pattern consists of four transactions, and each transaction is composed of 1.25 items.

4.2 Experimental Results

4.2.1 Performance Evaluation for Sequence Databases with Single-Item Events

Efficiency test. We first compared BIDE with CloSpan using the *Gazelle* data set. Fig. 4 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 percent to 0.01 percent. In Fig. 4, we see that many long closed sequences can be discovered for this sparse data set. For example, at support 0.01 percent, the longest frequent closed sequence has a length of 127. Figs. 6 and 7 demonstrate the runtime and memory usage comparison between BIDE and CloSpan. We can see that BIDE always runs much faster than CloSpan but consumes much less memory. For example, at support 0.01 percent, BIDE can be more than an order of magnitude faster than CloSpan, and it uses less memory by more than an order of magnitude.

Fig. 5 depicts the distribution of the number of frequent closed sequences against the length of the frequent closed sequences for data set *Snake*. We can see that it is a little

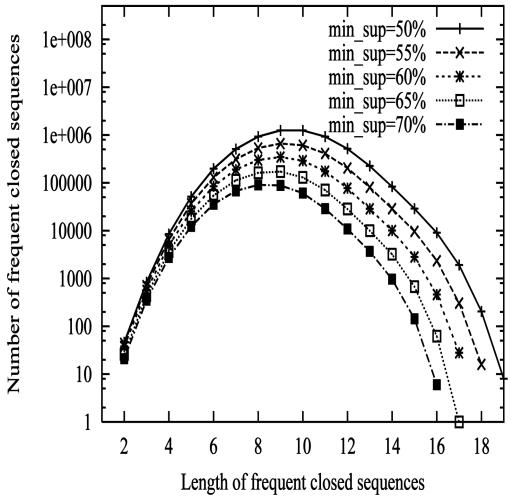

Fig. 5. Data set *Snake* (distribution).

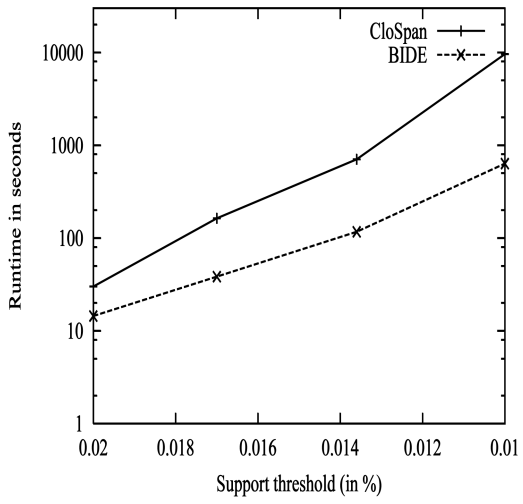

Fig. 6. Data set *Gazelle* (runtime).

slightly data set: A lot of closed sequences with a medium length from 6 to 12 can be mined with some not very low support thresholds from 50 percent to 70 percent. Fig. 8 shows that for this data set, BIDE is several times slower than CloSpan at a high support threshold, but, once the support is no greater than 60 percent, BIDE will significantly outperform CloSpan. For example, at support 50 percent, BIDE is about 40 times faster than CloSpan. In Fig. 9, we can see BIDE uses more than two orders of magnitude less memory than CloSpan in almost all the cases. This data set 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 300Mbytes of memory. For example, at support 50 percent, BIDE uses only about 2 Mbytes of memory, whereas CloSpan uses about 328 Mbytes of memory.

We also used the very dense data set *Pi* to compare BIDE with CloSpan. In Fig. 10, we can see that even with a very high support like 90 percent, there can be a large number of short frequent closed sequences with a length less than 10. Fig. 12 shows that with a support higher than 90 percent,

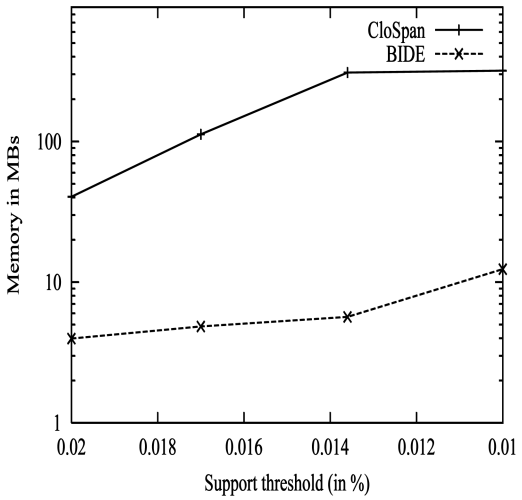

Fig. 7. Data set *Gazelle* (memory).

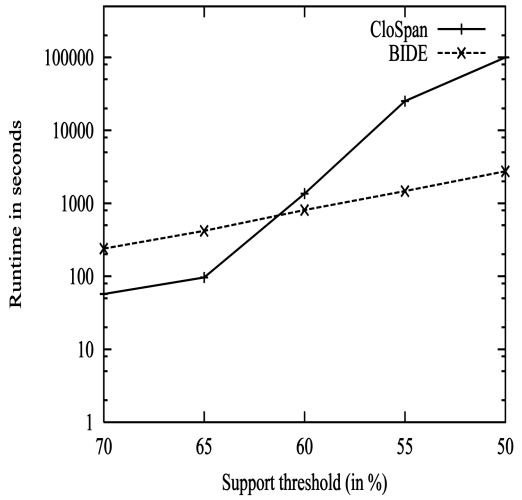

Fig. 8. Data set *Snake* (runtime).

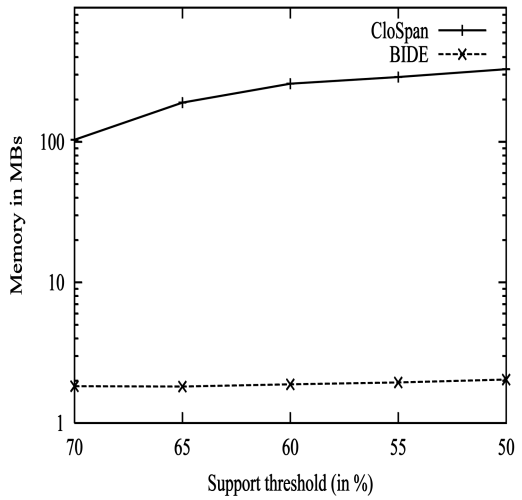

Fig. 9. Data set *Snake* (memory).

these two algorithms have very similar performance; CloSpan is only a little faster than BIDE, but once the support is no greater than 88 percent, BIDE will outperform CloSpan a lot. For example, at support 88 percent, BIDE is

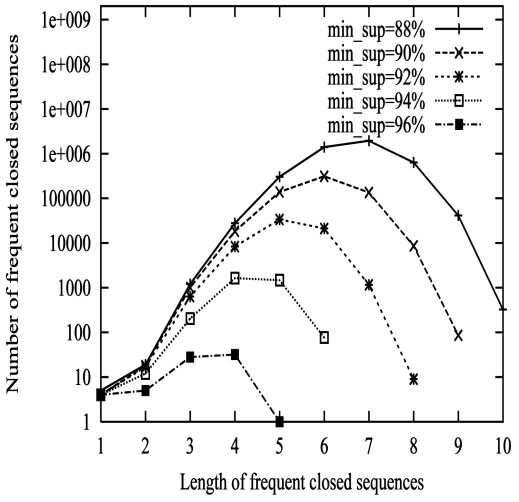

Fig. 10. Data set *Pi* (distribution).

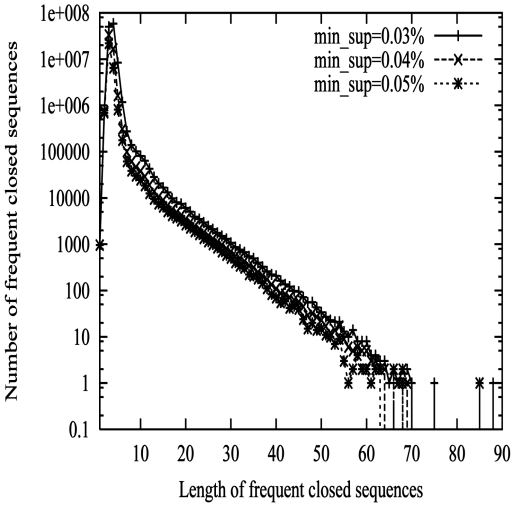

Fig. 11. Data set *C10T8S8I8* (distribution).

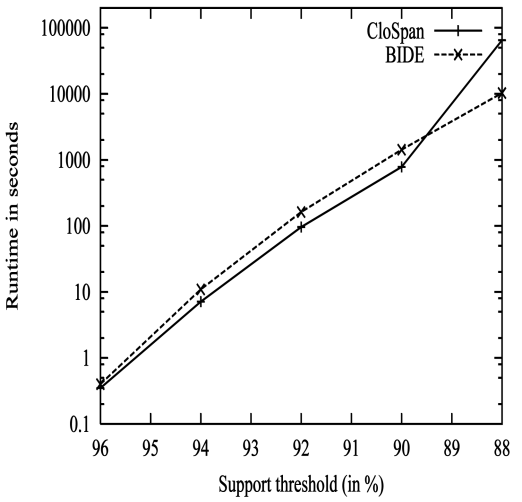

Fig. 12. Data set *Pi* (runtime).

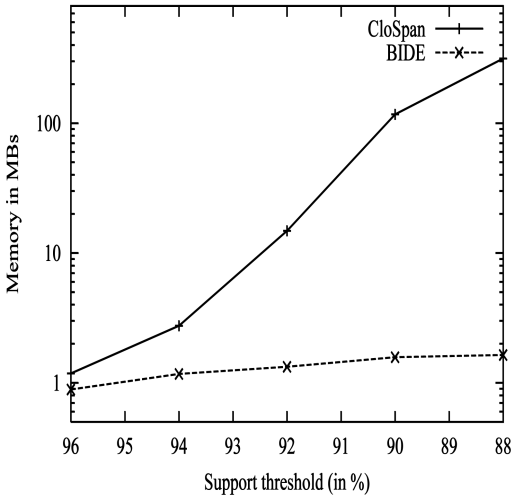

Fig. 13. Data set Pi (memory).

more than six times faster than CloSpan. In Fig. 13, we know BIDE always uses much less memory than CloSpan. At support 88 percent, BIDE consumes more than two orders of magnitude less memory than CloSpan.

Scalability test. We tested BIDE's scalability in both runtime and memory usage using all of the three real data sets in terms of the base size. In Figs. 14 and 15, we fixed the support threshold at a certain constant for each data set and replicated the sequences from 2 to 16 times. Although these three data sets have rather different features, BIDE shows a linear scalability in both the runtime and memory usage against the increasing number of sequences for these data sets. For example, for data set *Snake* with a given support of 60 percent, its runtime increases from 807 sec to 11,906 sec, and its memory usage increases from 1.883 Mbytes to 21.784 Mbytes when the number of sequences increases 16 times.

Effectiveness of the optimization technique in BIDE. Fig. 16 tests the effectiveness of the *BackScan* pruning method adopted in BIDE algorithm. We see that the *BackScan* pruning method is very effective in pruning the search space and speeding up the mining process: For the *Gazelle* data set with a support threshold of 0.02 percent, it can give several orders of magnitude enhancement to the performance. The effectiveness of the *BackScan* pruning method assures that BIDE, which is based only on this single pruning method, can significantly outperform in most cases the CloSpan algorithm, which adopts several pruning methods.

4.2.2 Performance Evaluation for Sequence Databases with Multiple-Item Events

Efficiency test. We used the real data set *Gazelle* and synthetic data sets *C10T8S8I8* and *C200T2.5S10I1.25* to test the efficiency of BIDE on general sequence databases.[3] For real data set *Gazelle*, here, we used its generalized version. As in [32] and [33], we treated each session in a Web click stream as an event and, thus, converted the Web click-stream data into a generalized form of sequences of subsets of items. We denote the generalized version of *Gazelle* by *Gazelle** in the following. Figs. 18 and 19 depict the

3. Note that the experiments regarding the performance test on general sequence databases were conducted on a DELL Inspiron 5100 machine with a 512-Mbyte memory and a 2.66-GHz CPU installed.

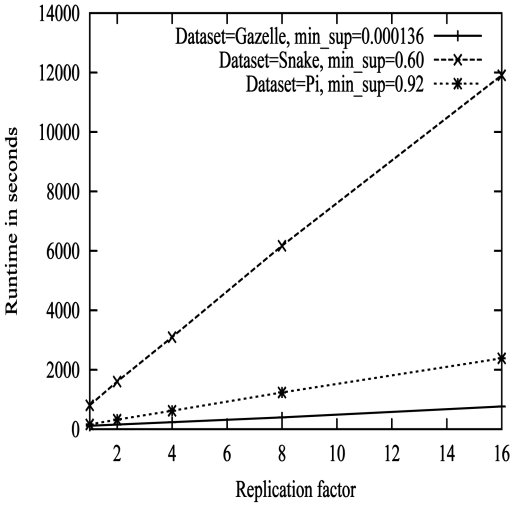

Fig. 14. Scalability test (single-item event, runtime).

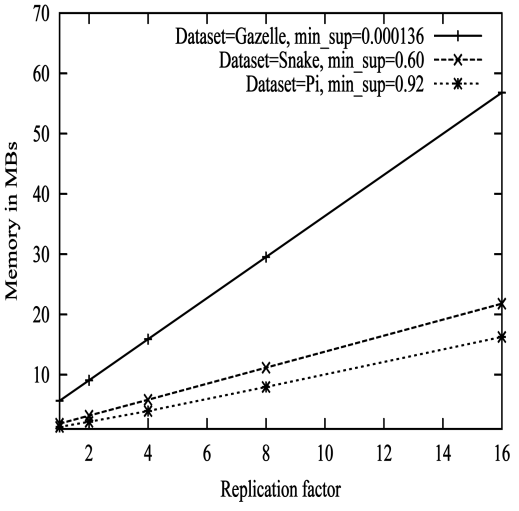

Fig. 15. Scalability test (single-item event, memory).

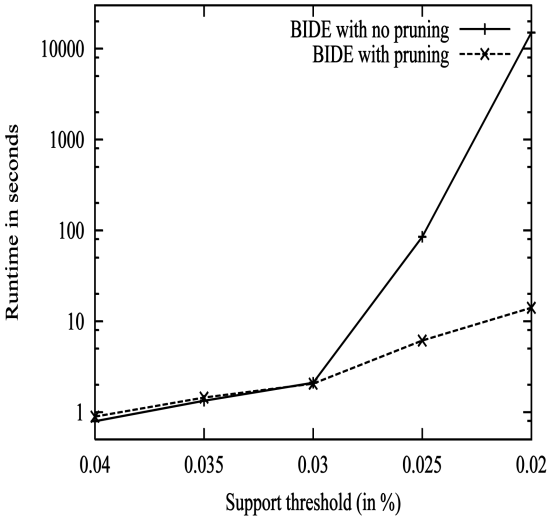

Fig. 16. Pruning test (single-item event, *Gazelle*).

comparison results between BIDE and CloSpan for data set *Gazelle**. At a high support, CloSpan is about two times faster than BIDE, but at a low support, it can be orders of

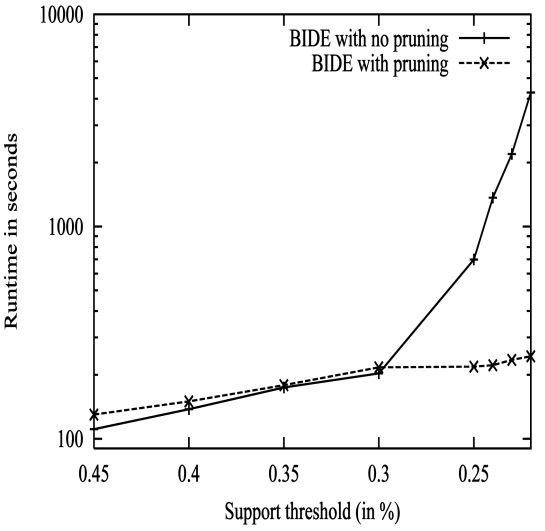

Fig. 17. Pruning test (multiple-item event, *C10T8S8I8*).

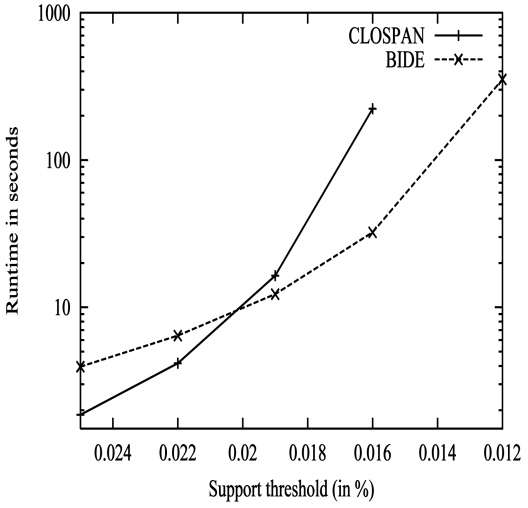

Fig. 18. Data set *Gazelle** (runtime).

magnitude slower than BIDE. In Fig. 19, we see that BIDE always consumes much less memory than CloSpan.

Fig. 11 shows the distribution of frequent closed sequences for data set *C10T8S8I8* when the support threshold is set at 0.05 percent, 0.04 percent, and 0.03 percent. We see that this data set is a challenging one, as it can yield some very long patterns. Fig. 20 shows that at a high support, CloSpan can be several times faster than BIDE, but once we lower the support to a certain point, BIDE becomes much more efficient than CloSpan, and sometimes, it can be several orders of magnitude faster than CloSpan. As to the memory usage, we see in Fig. 21 that BIDE consumes over an order of magnitude less memory than CloSpan for a wide range of support thresholds.

Figs. 22 and 23 depict the performance comparison results between BIDE and CloSpan for the relatively large data set *C200T2.5S10I1.25*. We see that similar to the results for the other two data sets, BIDE always uses much less memory than CloSpan. As to the runtime, CloSpan is several times faster than BIDE at a high support, but it can be orders of magnitude slower than BIDE at a low support.

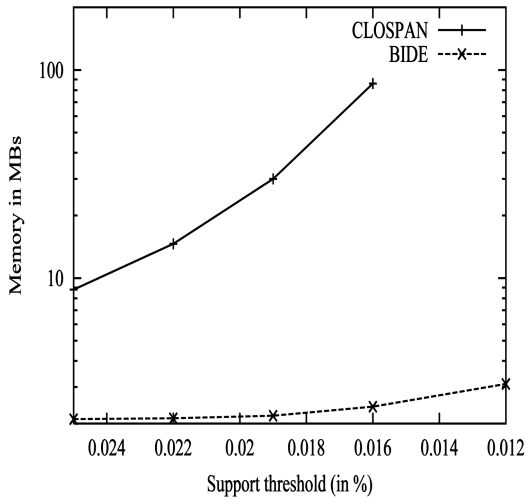

Fig. 19. Data set *Gazelle** (memory).

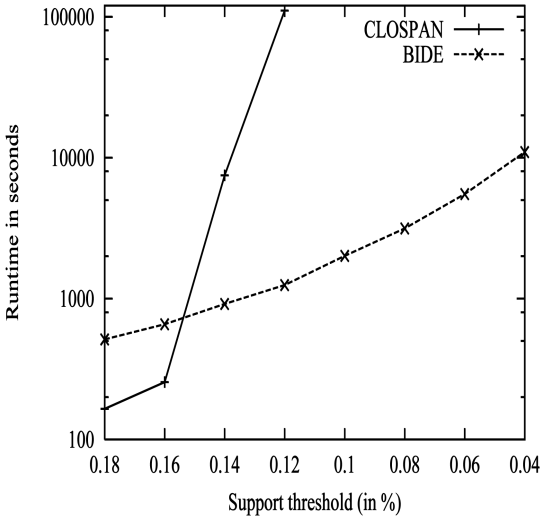

Fig. 20. Data set *C10T8S8I8* (runtime).

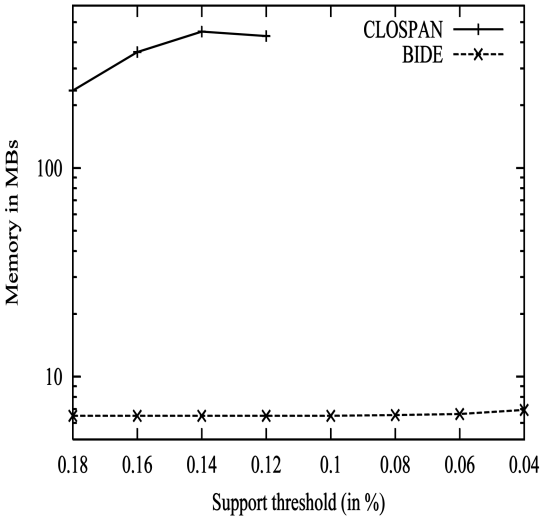

Fig. 21. Data set *C200T2.5S10I1.25* (memory).

Scalability test. We tested the scalability of BIDE against base size using the T2.5S10I1.25 data set series. In the experiments, we adopted three different support thresholds: 1 percent, 0.5 percent, and 0.25 percent. Fig. 24 shows the runtime scalability test results. The three curves in the figure correspond to the three different support settings. We see that the runtime of BIDE increases linearly with the increasing number of input sequences from 200,000 to 1,000,000. Fig. 25 shows the space usage scalability test. We see that BIDE also has good scalability for general sequence databases in terms of memory usage, and the difference between the space usage with regard to the three different support thresholds is very marginal.

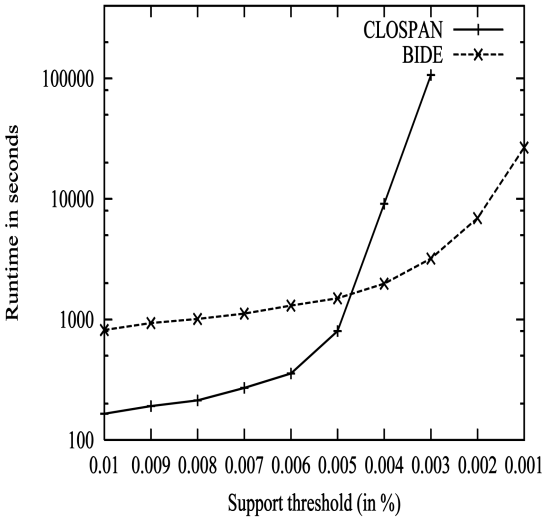

Fig. 22. Data set *C200T2.5S10I1.25* (runtime).

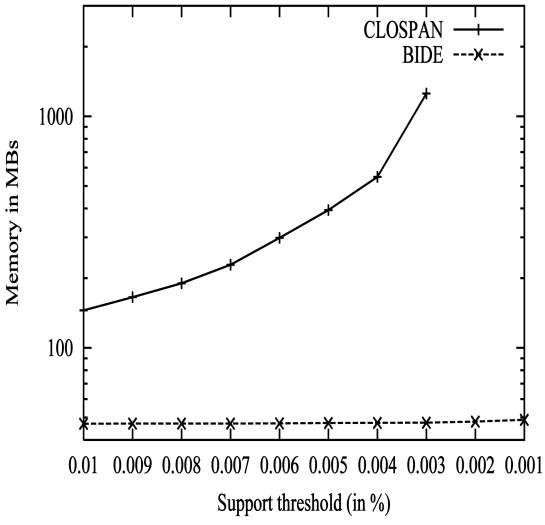

Fig. 23. Data set *C200T2.5S10I1.25* (memory).

Effectiveness of the optimization technique. The *BackScan* pruning method has also been generalized to be applied in the BIDE algorithm. We evaluated the effectiveness of this pruning method using the *C10T8S8I8* data set. In Fig. 17, we see that the *BackScan* pruning method is also very effective in improving the efficiency of the BIDE algorithm with general sequence databases.

5 Conclusions

Many studies have elaborated that closed pattern mining has the same expressive power as that of all frequent

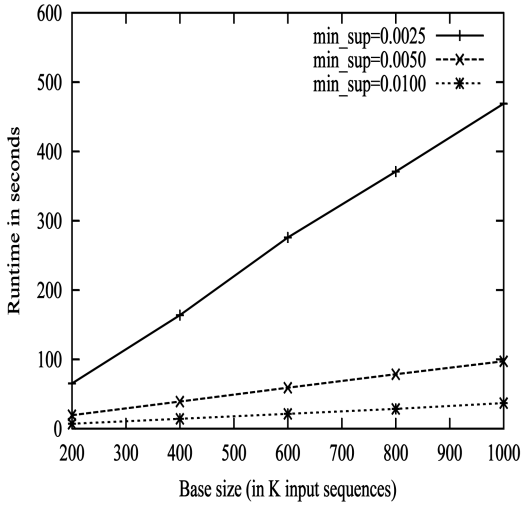

Fig. 24. Scalability test (*T2.5S10I1.25*, runtime).

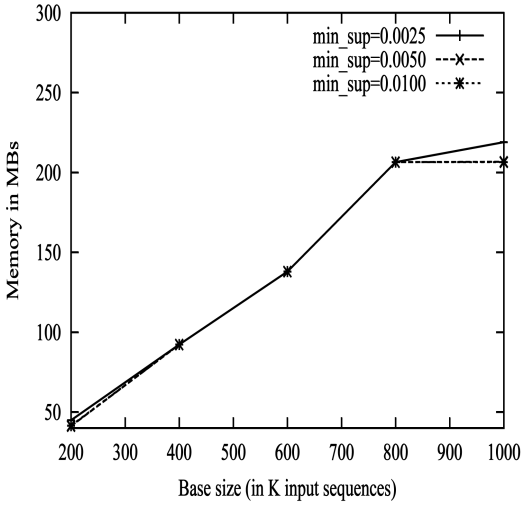

Fig. 25. Scalability test (*T2.5S10I1.25*, memory).

pattern mining yet leads to a more compact result set and significantly better efficiency. 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, it will not only 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 historical 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. In addition, BIDE algorithm can be easily adapted to mine frequent closed sequences of subsets of items. 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 a more compact and specific result set.

ACKNOWLEDGMENTS

The authors are grateful to Dr. Mohammed Zaki at Rensselaer Polytechnic Institute for providing them with the source code of SPADE, Dr. George Karypis at the University of Minnesota for providing them with the source of some protein sequence data, and Dr. Xifeng Yan at the University of Illinois for providing them with the executable code of CloSpan and some helpful discussions. The research of Jianyong Wang and Chun Li was supported in part by 973 Program under Grant 2006CB303103, the Basic Research Foundation of Tsinghua National Laboratory for Information Science and Technology (TNList), the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry of China, and the National Natural Science Foundation of China (NSFC) under Grant 60573061. The research of Jiawei Han was supported in part by the US National Science Foundation NSF IIS-05-13678/06-42771 and NSF BDI-05-15813. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies. A preliminary conference version of this paper appeared in the *Proceedings of the 2004 International Conference on Data Engineering (ICDE '04)*.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. Int'l Conf. Very Large Data Bases (VLDB '94)*, pp. 487-499, Sept. 1994.

[2] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. Int'l Conf. Data Eng. (ICDE '95)*, pp. 3-14, 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," *J. Molecular Biology*, vol. 311, pp. 395-408, 2001.

[4] J. Ayres, J. Gehrke, T. Yiu, and J. Flannick, "Sequential Pattern Mining Using a Bitmap Representation," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD '02)*, pp. 429-435, July 2002.

[5] *Proc. ICDM Workshop Frequent Itemset Mining Implementations (FIMI '04)*, R.J. Bayardo Jr., B. Goethals, and M.J. Zaki, eds., Nov. 2004.

[6] G. Casas-Garriga, "Summarizing Sequential Data with Closed Partial Orders," *Proc. SIAM Int'l Conf. Data Mining (SDM '05)*, Apr. 2005.

[7] S. Cong, J. Han, and D.A. Padua, "Parallel Mining of Closed Sequential Patterns," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD '05)*, pp. 562-567, Aug. 2003.

[8] M. Deshpande and G. Karypis, "Evaluation of Techniques for Classifying Biological Sequences," *Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD '02)*, pp. 417-431, May 2002.

[9] M. Garofalakis, R. Rastogi, and K. Shim, "SPIRIT: Sequential Pattern Mining with Regular Expression Constraints," *Proc. Int'l Conf. Very Large Data Bases (VLDB '99)*, pp. 223-234, Sept. 1999.

[10] J. Han, J. Wang, Y. Lu, and P. Tzvetkov, "Mining Top-K Frequent Closed Patterns without Minimum Support," *Proc. IEEE Int'l Conf. Data Mining (ICDM '02),* pp. 211-218, Dec. 2002.
[11] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.C. Hsu, "FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD '00),* pp. 355-359, Aug. 2000.
[12] X. Ji, J. Bailey, and G. Dong, "Mining Minimal Distinguishing Subsequence Patterns with Gap Constraints," *Proc. IEEE Int'l Conf. Data Mining (ICDM '05),* pp. 194-201, Nov. 2005.
[13] I. Jonassen, J.F. Collins, and D.G. Higgins, "Finding Flexible Patterns in Unaligned Protein Sequences," *Protein Science,* vol. 4, no. 8, pp. 1587-1595, 1995.
[14] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng, "KDD-Cup 2000 Organizers' Report: Peeling the Onion," *SIGKDD Explorations,* vol. 2, pp. 86-98, 2000.
[15] Z. Li, Z. Chen, S. Srinivasan, and Y. Zhou, "C-Miner: Mining Block Correlations in Storage Systems," *Proc. Usenix Conf. File and Storage Technologies (FAST '04),* pp. 173-186, Mar. 2004.
[16] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code," *IEEE Trans. Software Eng.,* vol. 32, no. 3, pp. 176-192, Mar. 2006.
[17] H. Mannila, H. Toivonen, and A.I. Verkamo, "Discovering Frequent Episodes in Sequences," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD '95),* pp. 210-215, Aug. 1995.
[18] F. Masseglia, F. Cathala, and P. Poncelet, "The PSP Approach for Mining Sequential Patterns," *Proc. European Symp. Principle of Data Mining and Knowledge Discovery (PKDD '98),* pp. 176-184, Sept. 1998.
[19] B. Ozden, S. Ramaswamy, and A. Silberschatz, "Cyclic Association Rules," *Proc. Int'l Conf. Data Eng. (ICDE '98),* pp. 412-421, Feb. 1998.
[20] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering Frequent Closed Itemsets for Association Rules," *Proc. Int'l Conf. Database Theory (ICDT '99),* pp. 398-416, Jan. 1999.
[21] J. Pei, J. Han, and R. Mao, "CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets," *Proc. ACM SIGMOD Workshop Research Issues in Data Mining and Knowledge Discovery (DMKD '00),* pp. 21-30, May 2000.
[22] 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 Trans. Knowledge and Data Eng.,* vol. 16, no. 11, pp. 1424-1440, Nov. 2004.
[23] J. Pei, J. Han, and W. Wang, "Constraint-Based Sequential Pattern Mining in Large Databases," *Proc. Int'l Conf. Information and Knowledge Management (CIKM '02),* pp. 18-25, Nov. 2002.
[24] J. Pei, J. Liu, H. Wang, K. Wang, P.S. Yu, and J. Wang, "Efficiently Mining Frequent Closed Partial Orders," *Proc. IEEE Int'l Conf. Data Mining (ICDM '05),* pp. 753-756, Nov. 2005.
[25] M. Seno and G. Karypis, "SLPMiner: An Algorithm for Finding Frequent Sequential Patterns Using Length-Decreasing Support Constraint," *Proc. IEEE Int'l Conf. Data Mining (ICDM '02),* pp. 418-425, Dec. 2002.
[26] R. She, F. Chen, K. Wang, M. Ester, J.L. Gardy, and F.S.L. Brinkman, "Frequent-Subsequence-Based Prediction of Outer Membrane Proteins," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD '03),* pp. 436-445, Aug. 2003.
[27] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," *Proc. Int'l Conf. Extending Database Technology (EDBT '96),* pp. 3-17, Mar. 1996.
[28] P. Tzvetkov, X. Yan, and J. Han, "TSP: Mining Top-K Closed Sequential Patterns," *Proc. IEEE Int'l Conf. Data Mining (ICDM '03),* pp. 347-354, Dec. 2003.
[29] J. Wang, J. Han, and J. Pei, "CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD '03),* pp. 236-245, Aug. 2003.
[30] T. Xie and J. Pei, "MAPO: Mining API Usages from Open Source Repositories," *Proc. Third Int'l Workshop Mining Software Repositories (MSR '06),* pp. 54-57, May 2006.
[31] T. Xie and J. Pei, "Data Mining for Software Engineering," *Proc. 2006 ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD '06),* tutorial, Aug. 2006.
[32] X. Yan, J. Han, and R. Afshar, "CloSpan: Mining Closed Sequential Patterns in Large Databases," *Proc. SIAM Int'l Conf. Data Mining (SDM '03),* pp. 166-177, May 2003.
[33] J. Yang, P.S. Yu, W. Wang, and J. Han, "Mining Long Sequential Patterns in a Noisy Environment," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '02),* pp. 406-417, June 2002.
[34] M. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences," *Machine Learning,* vol. 42, pp. 31-60, 2001.
[35] M. Zaki and C. Hsiao, "CHARM: An Efficient Algorithm for Closed Itemset Mining," *Proc. SIAM Int'l Conf. Data Mining (SDM '02),* pp. 457-473, Apr. 2002.

Jianyong Wang received the PhD degree in computer science in 1999 from the Institute of Computing Technology, the Chinese Academy of Sciences. Since then, he has worked as an assistant professor in the Department of Computer Science and Technology, Peking University, in the areas of distributed systems and Web search engines and visited the School of Computing Science at Simon Fraser University, the Department of Computer Science at the University of Illinois at Urbana-Champaign, and the Digital Technology Center and Department of Computer Science and Engineering at the University of Minnesota, mainly working in the area of data mining. He is currently an associate professor in the Department of Computer Science and Technology, Tsinghua University, Beijing. He is a senior member of the IEEE and a member of the ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD).

Jiawei Han is a professor in the Department of Computer Science, University of Illinois at Urbana-Champaign. He has been working on research into data mining, data warehousing, stream data mining, spatiotemporal and multimedia data mining, biological data mining, social network analysis, text and Web mining, and software bug mining, with more than 300 conference and journal publications. He has chaired or served on many program committees of international conferences and workshops. He also served or is serving on the editorial boards of *Data Mining and Knowledge Discovery*, the *IEEE Transactions on Knowledge and Data Engineering*, the *Journal of Computer Science and Technology*, and the *Journal of Intelligent Information Systems*. He is currently the founding editor-in-chief of the *ACM Transactions on Knowledge Discovery from Data*, and on the board of directors for the Executive Committee of ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD). He is an ACM fellow and an IEEE senior member and received the ACM SIGKDD Innovation Award (2004) and IEEE Computer Society Technical Achievement Award (2005).

Chun Li received the bachelor's degree in computer science from the Department of Computer Science and Technology, Tsinghua University, Beijing, in 2006. He is currently a graduate student in the same department at Tsinghua University. His research interests mainly include data mining and knowledge discovery, especially sequential pattern mining and its applications on bio-data.