Knowledge-Based Systems 89 (2015) 1–13

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

CCSpan: Mining closed contiguous sequential patterns

Jingsong Zhang [a], Yinglin Wang [b,*], Dingyu Yang [a]

[a] Dept. of CSE, Shanghai Jiao Tong University, Shanghai, China
[b] Dept. of CST, Shanghai University of Finance and Economics, Shanghai, China

ARTICLE INFO

Article history:
Received 21 March 2015
Received in revised form 20 June 2015
Accepted 24 June 2015
Available online 2 July 2015

Keywords:
Data mining
Sequential pattern mining
Closed sequential pattern
Contiguous constraint
Closed contiguous sequential pattern

ABSTRACT

Existing closed sequential pattern mining generates a more compact yet complete resulting set compared with general sequential pattern mining. However, conventional closed sequential pattern mining algorithms pose a great challenge at spawning a large number of inefficient and redundant patterns, especially when using low support thresholds or pattern-enriched databases. Driven by wide applications of sequential patterns with contiguous constraint, we propose CCSpan (Closed Contiguous Sequential pattern mining), an efficient algorithm for mining closed contiguous sequential patterns, which contributes to a much more compact pattern set but with the same information w.r.t. closed sequential patterns. Moreover, with the shorter feature of patterns, the closed contiguous sequential patterns are preferred for feature selection and sequence classification based on the Minimum Description Length principle. CCSpan adopts a novel snippet-growth paradigm to generate a series of snippets as candidates, each of which is attached with a set of item(s) that precisely record the pattern's occurrences in the database, and CCSpan leverages three pruning techniques to improve the computational efficiency significantly. Our experiments based on both sparse and dense datasets demonstrated that CCSpan is efficient and scalable in terms of both database size and support threshold.

1. Introduction

Sequential patterns can provide useful information that frequent itemsets alone are not able to provide. Sequential pattern (SP) mining, which discovers frequent subsequences as patterns in sequence databases, is an important data mining problem with broad applications, such as feature selection for sequence classification and prediction [1–3], discovering access patterns in Web logs [4], finding copy-paste and related bugs in software code [5], and biological sequence analysis [6,7]. Many sequential pattern mining approaches have been studied, such as general sequential pattern mining [8,9], closed sequential pattern (CloSP) mining [10–13], maximal sequential pattern mining [14–16], and interesting sequential pattern mining [17–22]. In particular, closed sequential pattern mining has become an active topic in data mining community, since it is a compact yet lossless compression of sequential patterns.

Although some typical algorithms, such as CloSpan [10], BIDE [11], ClaSP [12] and CM-ClaSP [13], have been developed for mining closed sequential patterns, such algorithms often generate a large number of frequent patterns satisfying the support threshold, especially when the support threshold is low or the database is rich in frequent patterns. For example, in biology domain, the set of closed sequential patterns derived by previous mining methods has a significantly greater size than the corresponding database. Such derived set is too huge to be used effectively, which is an open issue of previous pattern mining algorithms. Moreover, to generate these patterns, the cost of mining process is prohibitively expensive. Traditional (closed) sequential pattern mining algorithms, including Apriori-based and pattern-growth, usually suffer from a poor scalability in terms of support threshold and database density, because a large number of frequent patterns occupy considerable memory and give rise to a huge search space for closure checking of new patterns or pattern growth, especially when the low support threshold or the dense database is used. Also, such a huge number of closed sequential patterns make some further data analysis like feature selection and classification a very challenging task. Hence, we aim at the ideal sequential patterns that are as compact as possible and meanwhile carry the same information w.r.t. the closed sequential patterns.

Fortunately, not all frequent sequential patterns are interesting for stakeholders. Some sequential patterns meeting a/some specific constraint(s) have attracted more attention, since they lead to much fewer patterns and have a better overall classification performance reported in [23]. One example is the closed contiguous

* Corresponding author. Tel.: +86 13818009080; fax: +86 2160279432.
E-mail addresses: jasun_zhang@163.com (J. Zhang), yinglin.wang@outlook.com (Y. Wang), yangdingyu8686@sjtu.edu.cn (D. Yang).

http://dx.doi.org/10.1016/j.knosys.2015.06.014
0950-7051/

sequential pattern, in which the items appearing in the sequences that contain the pattern must be adjacent w.r.t. the underlying ordering as typically defined in the closed sequential pattern. Using sequential patterns with contiguous constraint greatly benefit a wide spectrum of real-life tasks. There are several typical examples: (1) Web log mining [24,25]: For pre-fetching and caching, knowledge of such ordered contiguous page references is useful for predicting future references; furthermore, knowledge of frequent backward traversal is useful for improving the design of web pages. (2) DNA and amino acid sequence analysis [26,27]: Frequent contiguous sequential patterns in biological sequences often reflect meaningful features. Finding such patterns is essential to unveil common shared functions. (3) Mining IO traces [28,29]: The analysis of contiguous IO access patterns is beneficial to optimize the design of softwares, and thus improve the robustness of systems. In addition, based on a common sequence database, the average length of the closed contiguous sequential patterns is much shorter than that of general closed sequential patterns with the same support threshold. Therefore, the former patterns are usually preferable to the latter ones according to the Minimum Description Length (MDL) principle [28,29], which has a sound statistical foundation rooted in the well-known Bayesian inference and Kolmogorov complexity.

Previous closed sequential pattern mining algorithms including CloSpan [10], BIDE [11], ClaSP [12] and CM-ClaSP [13], have focused on the scalability of general closed sequential pattern mining, which does not involve the contiguous constraint of patterns. Such techniques cannot be directly applied to the closed contiguous sequential pattern mining. This is because the set of closed contiguous sequential patterns is not a proper subset of the set of closed sequential patterns or general sequential patterns. Existing methods using a post-pruning step do not perform the closed contiguous sequential pattern mining. Moreover, such methods do not record the patterns occurrences in the database. By pushing the contiguous constraint (i.e., pre-pruning) into the mining process, they will consume more memory space and runtime. Consequently, the major challenge is how to design an effective algorithm to ensure the resulting patterns are contiguous and closed. In the sequels, we will call the sequential patterns satisfying the contiguous yet closed constraints *closed contiguous sequential patterns*, and seek to mine them in an effective and efficient fashion.

To the best of our knowledge, we are the first to discover closed contiguous sequential patterns that reflect the adjacency of items and the closure of patterns simultaneously, which form a more compact resulting set but do not lose any information. The contributions of this piece of work include the following:

- We formalize the problem of mining closed contiguous sequential patterns, explore their properties, and present the first closed contiguous sequential pattern mining algorithm, CCSpan. Such algorithm and related datasets are available at our public website, http://cit.sjtu.edu.cn/CCSpan.aspx.
- We introduce a novel snippet-growth scheme to generate potential patterns. By splitting the original sequences into a set of snippets using the n-gram model, the proposed scheme suffices to guarantee the items of each snippet strictly keep the initial adjacency and ordering.
- We devise three skillful pruning techniques to greatly reduce the unpromising parts of search space, and utilize the transitivity of closed patterns to decompose the closure checking problem, and thus improve the efficiency of the CCSpan algorithm.
- Our experiments on both sparse and dense datasets demonstrate that, CCSpan outperforms other methods significantly regarding both effectiveness and efficiency. Moreover, CCSpan scales well in runtime and memory usage with the base size of synthetic datasets.

The remainder of the paper is organized as follows: Section 2 introduces the formulation of closed contiguous sequential pattern mining problem as well as some notations used throughout the paper. Following that, in Section 3, some properties of closed contiguous sequential patterns are first explored, and then our solution is presented step by step. Furthermore, we use an example to illustrate the execution traces of our algorithm in detail and analyze the time complexity of the problem. In Section 4 we illustrate an extensive experimental study. Section 5 reviews the related work. Finally, we conclude the study in Section 6.

2. Preliminary concepts

This section defines the basic concepts in sequential pattern mining, and then formally introduces the problem of closed contiguous sequential pattern mining.

An item is a basic unit that is of the minimal granularity in data mining. Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of distinct items. A sequence $S = \langle a_1, a_2, \ldots, a_n\rangle$ is an ordered list, where $a_i \in I$ is an item for $1 \leqslant i \leqslant n$. A sequence, for brevity, is also expressed as $s = a_1 a_2 \cdots a_n$. According to the definitions, note that an item a_i ($i \in [1, n]$) can occur multiple times in a sequence s. The size of a sequence, denoted as $|S|$, is the number of distinct items in the sequence. The length of a sequence, denoted as $l(S)$, is the total number of items including repeated one(s) in the sequence, i.e. $l(S) = n$. A sequence with k items is also termed a length-k sequence or k-sequence.[1] The number of distinct items in such a k-sequence is less than or equal to k. For example, one sequence *ABCFAC* is a 6-sequence while its size is 4. Aside from the above notations, we further present some more definitions, which provide a necessary background for the understanding of our subsequent algorithm.

Definition 1. Given two sequences $S_1 = \langle a_1 a_2 \cdots a_i\rangle$ and $S_2 = \langle b_1 b_2 \cdots b_j\rangle, S_1$ is a contiguous subsequence of S_2, denoted as $S_1 \sqsubseteq S_2$ (if $S_1 \neq S_2$, written as $S_1 \sqsubset S_2$), if and only if there exist integers $k_1, k_2, \ldots, k_i$ such that: (1) $1 \leqslant k_1 < k_2 < \cdots < k_i \leqslant j$; and (2) $a_1 = b_{k_1}, a_2 = b_{k_2}, \ldots, a_i = b_{k_i}$. We also call S_1 a snippet of S_2, S_2 a super-sequence of S_1, and S_2 contains S_1.

Definition 2. Given a sequence s and a sequence database D, the absolute support of s in D is the number of sequences in D that contain s, i.e., $Sup^a_D(s) = |\{S|S \in D \wedge s \sqsubseteq S\}|$. Similarly, the relative support of s in D is the proportion of sequences in D that contain s, i.e., $Sup^r_D(s) = |\{S|S \in D \wedge s \sqsubseteq\}|/|D|$.

In what follows, we use support $Sup_D(s)$ instead of both absolute support $Sup^a_D(s)$ and relative support $Sup^r_D(s)$ for simplicity.

Definition 3. Given a threshold σ, a contiguous subsequence s is a contiguous sequential pattern (ConSP) in database D if $Sup_D(s) \geqslant \sigma$.

Definition 4. Given a contiguous sequential pattern s in sequence database D, s is a closed contiguous sequential pattern (CloConSP) if there exists no contiguous sequential pattern s' such that: (1) $s \sqsubset s'$, and (2) $Sup_D(s) = Sup_D(s')$.

Definition 5. Given two sequences $s_1 = \langle a_1 a_2 \cdots a_i\rangle$ and $s_2 = \langle b_1 b_2 \cdots b_j\rangle$, and a sequence database D, s_1 is absorbed by s_2 (or s_2 absorbs s_1), denoted as $s_1 \sqsubset_= s_2$, if (1) $l(s_1) \geqslant 1, s_1 \sqsubset s_2$, and (2) $Sup_D(s_1) = Sup_D(s_2)$.

[1] K-subsequence and k-pattern are used to indicate the same meanings.

Definition 6. Provided two sequences $s_1 = \langle a_1 a_2 \cdots a_i \rangle$ and $s_2 = \langle b_1 b_2 \cdots b_j \rangle, s_1$ is a pre-subsequence of s_2, denoted as $s_1 \sqsubset_{pre} s_2$, if (1) $l(s_1) \geqslant 1, l(s_2) - l(s_1) = 1$; and (2) $a_1 = b_1, a_2 = b_2, \ldots, a_i = b_{j-1}$. Similarly, s_1 is a post-subsequence of s_2, denoted as $s_1 \sqsubset_{post} s_2$, if (1) $l(s_1) \geqslant 1, l(s_2) - l(s_1) = 1$; and (2) $a_1 = b_2, a_2 = b_3, \ldots, a_i = b_j$. Pre-subsequence and post-subsequence are collectively called pre-post-subsequence.

We highlight that a contiguous subsequence s may appear more than once in a sequence S in database D. In this case we only count once in accordance with the convention of previously developed sequential pattern mining algorithms. For convenience, we interchangeably use sequential pattern and SP, closed sequential pattern and CloSP, contiguous sequential pattern and ConSP, closed contiguous sequential pattern and CloConSP in the remainder of the paper.

Problem statement. Given a sequence database D and a minimum support threshold σ, the problem of mining closed contiguous sequential patterns is to discover the complete set F of frequent closed contiguous subsequences.

Example 1. Suppose that we have a sequence database D in Table 1 sharing with [11,30] and an absolute support $\sigma = 2$. Given a sequence database D sharing with [11,30] as shown in Table 1 and an absolute support $\sigma = 2$. The input database lists three distinct items and four input sequences (i.e. $|D| = 4$). Four types of frequent sequences, including frequent sequential pattern (F_{SP}), contiguous sequential pattern (F_{ConSP}), closed sequential pattern (F_{CloSP}), and closed contiguous sequential pattern ($F_{CloConSP}$), with their absolute supports attached after ":", are elicited from D respectively (see Table 2). The entire set of frequent sequences contains 17 patterns, while the full set of contiguous sequential patterns is only 7. Similarly, the whole set of closed sequential patterns consists of 6 patterns. In contrast, the size of closed contiguous sequential pattern set is only 4. Informally, given an input sequence database and a user-specified support threshold, the size of closed contiguous sequential pattern set is much smaller than that of closed sequential pattern set.

3. Efficient mining of closed contiguous sequential patterns

Before diving into developing a new CCSpan algorithm, we consider the following questions: (1) How do we obtain the potential patterns given the fact that they must strictly maintain the original adjacency of items? (2) Upon getting a candidate subsequence, how do we design search space pruning scheme(s) to accelerate the mining process? (3) How do we check whether a contiguous sequential pattern is closed or not? In the sequels, we elaborate three steps of CCSpan and answer these three questions.

3.1. CCSpan overview

The contiguous and closed constraints along with the properties of sequential patterns inspire a three-step design for CCSpan. First, each sequence S of input sequence database D is split into a series of snippets (i.e., candidate contiguous subsequences) which preserve the original ordering of items. The length of the snippets is fixed within one complete scan in D, while it incrementally increases by step length 1 according to the n-gram model [31] in subsequent scans. Second, by exploring some properties of closed contiguous sequential patterns, we develop three effective pruning techniques, checked snippet pruning, pro-post-subsequence pruning, and support pruning to prune the pointless parts of search space. The occurrence frequencies of the remaining candidates, which passed through the first two pruning procedures will be counted in the database respectively. We iterate this process until no more distinct patterns exist. A full set consisting of all contiguous sequential patterns is obtained accordingly. In the third step, all non-closed contiguous sequential patterns are distinguished successively from the set of contiguous sequential patterns by performing the closure checking, as suggested in Definition 4 and Theorem 2.

Table 1
An example sequence database D.

Sequence Id	Sequence
1	CAABC
2	ABCB
3	CABC
4	ABBCA

Table 2
Comparison of four types of frequent sequences.

Sequence type	Frequent sequences
F_{SP}	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
F_{ConSP}	C:4, A:4, B:4, CA:3, AB:4, BC:4, ABC:3
F_{CloSP}	AA:2, ABB:2, ABC:4, CA:3, CABC:2, CB:3
$F_{CloConSP}$	CA:3, AB:4, BC:4, ABC:3

3.2. Candidate generation

Due to the lack of consideration on the specific constraint(s) of sequential patterns, many conventional algorithms developed for mining closed sequential patterns need to enumerate all possible combinations of frequent subsequences to produce potential longer patterns, rendering both memory and runtime consumption intractable. In addition, two adjacent elements (or items) in a pattern (see F_{SP} and F_{CloSP}) may not be adjacent in a sequence that contains the pattern, which is unsuitable for some tasks as mentioned earlier. To tackle such two problems, we develop a snippet-growth scheme to generate candidates, the items of which strictly keep the original adjacency and ordering. Every candidate is produced by splitting the sequences of input database rather than by enumerating all possible joints of elements of a seed set.

Given a sequence database D, let S be a sequence in D. In the first stage, called C_1-splitting, each sequence S is split into a set of length-1 snippets (subsequences), in which each snippet contains only a single item. And these snippets are checked if they are frequent. In the second stage, called C_2-splitting accordingly, each sequence S is also split into some snippets by length-2 window and the frequencies of them are determined in next pruning steps. Unlike the candidates through classic algorithms, such snippets (candidates) are both under a contiguous constraint and preserving the original ordering in database. The following stages are similar to the second one. Such a process repeats until no more candidates generate or no candidates equal to or exceed the minimum support. Here, we draw a concrete example to illustrate the workings of the candidate generation.

Example 2. The first sequence of Table 1, i.e., $S_1 = \langle CAABC \rangle$, is reused in this example. In C_1-splitting stage, a 1-subsequence set, i.e., $C_1 = \{C, A, A, B, C\}$[2] is formed by splitting the sequence S_1. And then the C_2-splitting generates a 2-subsequence set $C_2 = \{CA, AA, AB, BC\}$ that is contains four elements (subsequences). The remaining

[2] The repeated elements such as items A and C, will be first discarded in next pruning step.

sets produced by CCSpan are $C_3 = \{CAA, AAB, ABC\}, C_4 = \{CAAB, AABC\}$, and $C_5 = \{CAABC\}$ respectively.

Generally, the process stops its splitting far before the snippets actually reach the maximum length of sequences (for example, length-5 in the above example) by employing our pruning paradigms, which will be elaborated in next subsection.

3.3. Search space pruning

Every sequence of database is discretized into a serious of snippets as candidate subsequences which are rather coarse because they may contain many repeated and non-potential ones. To minimize the overhead of search space, we examine several characteristics of contiguous sequential patterns in the sequel, which underpin the design of pruning.

Assume that two sequences $s = \langle a_1 a_2 \cdots a_i \rangle$ and $s' = \langle b_1 b_2 \cdots b_j \rangle$, $s \sqcup s'$ means s concatenates s', i.e., $s \sqcup s' = \langle a_1 a_2 \cdots a_i b_1 b_2 \cdots b_j \rangle$. The pre-subsequence and post-subsequence of sequence s ($l(s) \geqslant 2$) are also given by

$$s_{pre} = f_{\sqsubset_{pre}}(s) = \bigsqcup_{i=0}^{l(s)-2} s_i \tag{1}$$

and

$$s_{post} = f_{\sqsubset_{post}}(s) = \bigsqcup_{i=1}^{l(s)-1} s_i \tag{2}$$

respectively, where i is the index of item in s.

Theorem 1. *Given a sequence s ($l(s) \geqslant 1$) and a sequence database D, suppose $Sup_D(s) = \sigma$, then each non-empty subsequence s' of s satisfies $Sup_D(s') \geqslant \sigma$.*

Proof. Let X be an itemset, X_{sub} be a set consisting of all subsets of X, T be a Transaction, suppose $Sup_T(X) = \sigma$, then each non-empty element $x \in X_{sub}$ satisfies $Sup_T(x) \geqslant \sigma$ according to the downward closure property of the Apriori. Without loss of generality, let Y be a sequence with all items of X listed by a certain ordering, Y_{sub} be a set consisting of all subsequences of Y, then the support of Y in T is the same as the X in T, i.e., $Sup_T(Y) = Sup_T(X) = \sigma$, while $Y_{sub} \subset X_{sub}$, then each non-empty element, i.e., non-empty subsequence $y \in Y_{sub}$ satisfies $Sup_T(y) \geqslant \sigma$. The correctness of Theorem 1 then becomes immediate. □

Lemma 1. *Given a sequence s ($l(s) \geqslant 2$), suppose s is frequent in sequence database D, then both pre-subsequence s_{pre} and post-subsequence s_{post} are frequent in D.*

Proof. Let F be a set consisting of all subsequences of s, then each element of F, i.e., $\forall s' \in F$ is frequent by virtue of Theorem 1. The pre-subsequence $s_{pre} \in F$ and the post-subsequence $s_{post} \in F$. Then both of them are frequent. The lemma holds immediately. □

Lemma 2. *Given a sequence s ($l(s) \geqslant 2$) and a sequence database D, if there exists no frequent pre-subsequence or post-subsequence of s, i.e., $Sup_D(s_{pre}) \ngeqslant \sigma$ or $Sup_D(s_{post}) \ngeqslant \sigma$ holds, then s can be safely pruned.*

Proof. Easily derived from Theorem 1 and Lemma 1. □

The pruning process of CCSpan is divided into three stages, which are successively elaborate in the following:

- *Checked snippet pruning.* In real-world datasets, some snippets often appear multiple times not only among different sequences but inside one sequence. For each newly split snippet, we check the previous snippets to see whether the new one already exists. The repeated snippets can be easily identified and discarded on-the-fly. Hence, it is desirable that the checked snippet pruning is assigned in the first step to ensure the repeated snippets are pruned as early as possible, which avoids unnecessary pro-post-subsequence checking and support counting. In short, this step leverages a repeated snippet checking strategy to guarantee any distinct snippet is detected only once which can speed up CCSpan's mining process.
- *Pre-post-subsequence pruning.* Unlike conventional candidate *enumerate and test* paradigm, pre-post-subsequence pruning, for a new length-k snippet s, does not need to check all its $(k-1)$-subsequences if there exist an infrequent one. Instead, it checks only the frequency of pre-post-subsequences of s. The length-k snippet is pruned immediately if its pre-subsequence or post-subsequence is infrequent as indicated by Lemma 2. Regarding the worst case, most of the previously developed sequential pattern mining methods need to be detected $C_k^{k-1} = k$ times. In contrast, our pruning scheme takes at most two times. Thus, such scheme can efficiently filter the futile snippets from the first pruning stage. We find that this manner significantly improves the performance in our real experiments and makes the cost of such pruning nearly negligible compared with the aggregate running time.
- *Support pruning.* A snippet is called a promising candidate if it satisfies: (1) It is distinct compared with the previous snippets (already split snippets); and (2) both pre-subsequence and post-subsequence of the snippet are frequent. Such snippets satisfying the above twofold conditions can be shifted to count their supports to check whether they are frequent. For each promising candidate s, it is natural to use the conventual matching method to count the actual support, however, CCSpan does not need to check every sequence of database whether it contains s. Instead, it only checks such sequences from the next one w.r.t. the current sequence identifier to the end of the database, which effectively prevents redundant snippet matching operations to accelerate the counting process.

Such a set consisting of length-k contiguous sequential patterns is formed when no more new k-patterns appear by iterating above process. The complete set of length-$(k-1)$ closed contiguous sequential patterns can be discovered by performing the sequel *pattern closure checking* scenario.

3.4. Pattern closure checking

The subsequent task is to distinguish non-closed contiguous sequential patterns from the general contiguous sequential patterns. The problem is to check out for each pattern s, whether there is a super-pattern absorbing s (see Definition 5). The conventional methods compare each pattern with other patterns in the discovered set, which is costly due to its $O(N^2)$ complexity. An interesting finding is that the closed contiguous sequential patterns hold the transitivity (see Theorem 2) among pattern subsets, each of which consists of single-length frequent patterns. Thus we can decompose the problem of pattern-closure-checking in the full pattern set into checking in each such two subsets with single-length patterns, where the pattern length deviation of the two subsets is only one. Using the transitivity property as a criterion for pattern closure checking, our algorithm contributes to a great computational efficiency.

Theorem 2. *Given three sequences s_1, s_2 and s_3, where $s_1 \sqsubset s_2$ and $s_2 \sqsubset s_3$, if both $s_1 \sqsubset_= s_2$ and $s_2 \sqsubset_= s_3$ hold, then $s_1 \sqsubset_= s_3$ holds in the meantime.*

Proof. Let $Sup_D(s_1) = \sigma_1, Sup_D(s_2) = \sigma_2$ and $Sup_D(s_3) = \sigma_3$. (i) Because $s_1 \sqsubset s_2$ and $s_2 \sqsubset s_3$, we have $s_1 \sqsubset s_3$ by Definition 1; (ii) based on $s_1 \sqsubset_= s_2$ and $s_2 \sqsubset_= s_3$, then $\sigma_1 = \sigma_2$ and $\sigma_2 = \sigma_3$ hold, so $\sigma_1 = \sigma_3$. With (i), (ii) and Definition 5, we complete our proof. □

Without loss of generality, assume that there are two contiguous sequential pattern sets, a set F_{k-1} consisting of length-$(k-1)$ patterns and a set F_k consisting of length-k ones (where $k \geqslant 2$). A pattern s in F_{k-1} is closed if there exists no element of F_k which can absorb s with the same support. Specifically, each pair of pre-post-subsequences of the length-k contiguous sequential patterns are first calculated by Eqs. (1) and (2). In the following, the set of length-$(k-1)$ contiguous sequential patterns is scanned for checking whether there exists a pattern satisfying itself and its support count are respectively equal to the pre-post-subsequence and the support count of current k-pattern based on Definition 4. A length-$(k-1)$ contiguous sequential pattern is non-closed if the above twofold conditions are satisfied simultaneously. This length-$(k-1)$ checking is continued until every element of the length-$(k-1)$ set has been visited. From the whole mining process point of view, once a new set of length-k contiguous sequential patterns is formed completely, the pattern closure checking step can be conducted on-the-fly. By progressively checking, the whole non-closed contiguous sequential patterns are efficiently identified and the remains are closed ones when CCSpan completes its searching.

3.5. CCSpan algorithm

In this subsection, for elaborating CCSpan, we introduce several data structures first. Three compact data structures are employed for performing our mining task as follows: The input sequence database D is represented by a set of 2-tuples $(S.id, S)$, where $S.id$ is a sequence identifier and S a sequence itself. The closed contiguous sequential pattern and the non-closed one share the same data structure: a triple $(f, f.count, B)$, where f is a pattern itself, $f.count$ is the support count of f in D, and the last attribute variable "B" takes on the values "Y" and "N". Y indicates f holds the closed, while N the non-closed. The final output of the algorithm is a set F, which consists of all contiguous sequential patterns including closed and non-closed ones. The inside patterns of F can be organized into a form of set $\{\{F_1\}, \{F_2\}, \ldots, \{F_k\}\}$ consisting of k different partitions, each of which is a subset with single-length patterns.

Algorithm 1 sketches CCSpan algorithm that mines the set of closed contiguous sequential patterns. Given a transformed database D and a support threshold σ, we define global variables F and F_k to store all length (closed) contiguous sequential patterns and only length-k contiguous sequential patterns respectively. The frequent 1-sequence set F_1 is first derived by running the init-gen() function (subroutine). Such 1-patterns are fed to ConSP-gen() function for checking longer patterns. In each subsequent pass k, there are three steps: *candidate generation*, *search space pruning* and *pattern closure checking*. Variable P_k is local to each pass (line 2). It is a set for storing all the newly split snippets during current length splitting process. P_k, as a parameter, is employed by ConSP-gen() function, and it can greatly reduce the computational cost of unqualified candidates pruning. Each initial candidate subsequence is derived from the original sequence splitting in database (lines 3 and 4). Function ConSP-gen() is invoked for performing the checked snippet pruning, pre-post-subsequence pruning, and support pruning (line 5). It returns a length-specified set of contiguous sequential patterns depending on the passes of CCSpan algorithm. Subsequently, based on the intermediate output set F_k consisting of length-k contiguous sequential patterns just obtained and the intermediate output set F_{k-1} consisting of length-$(k-1)$ contiguous sequential patterns last obtained, all closed contiguous sequential patterns in F_{k-1} are distinguished from general contiguous sequential patterns by calling CloConSP-gen() effectively (line 8).

Algorithm 1. CCSpan(D, σ)

Input: sequence database D, support threshold σ
$F \leftarrow \emptyset$; // initialize F to store the CloConSPs
$F_k \leftarrow \emptyset$; // initialize F_k to store the length-k ConSPs
$F_1 \leftarrow$ init-gen(D, σ) // generate the frequent 1-sequences
1: **for** $(k = 2; F_{k-1} \neq \emptyset; k++)$ **do**
2: $P_k \leftarrow \emptyset$ // initialize P_k to store the checked sequences
3: **for** each sequence $S \in D$ and $l(S) \geqslant k$ **do**
4: **for** each contiguous subsequence $s \in S$ and $l(s) = k$ **do**
5: ConSP-gen$(D, s, F_{k-1}, P_k, S.id, \sigma)$; // generate the ConSPs
6: **end for**
7: **end for**
8: $F_{k-1} \leftarrow$ CloConSP-gen(F_{k-1}, F_k); // generate the CloConSPs
9: $F \leftarrow \cup_{k-1} F_{k-1}$;
10: **end for**
11: **Output:** $F \leftarrow \cup_k F_k$;

The output of Algorithm 1 is a pattern set $F = \{(f, f.count, B) | f.count \geqslant \sigma\}$. The full closed contiguous sequential patterns can be easily derived from such F via attribute B, i.e., $F_{CloConSP} = \{e | e \in F \wedge B = Y\}$. Better still, the complete set F, which consists of all contiguous sequential patterns including closed and non-closed ones, can be regarded as a byproduct along with our mining task. In the sequel, the preceding three functions will be discussed briefly.

Function init-gen(), as the first yet unique pass, is run for finding all the frequent length-1 patterns, which are fed to the discovery of longer patterns. These candidate 1-subsequences provided by splitting the initial sequences are excluded safely if they exist in the snippet set P_1, because all elements of it have been checked before (lines 3 and 4). For calculating the support of a promising candidate, it does not need to scan the whole database. Instead, it only compares with such sequences from the next sequence of current identifier to the last one in the database (lines 6–10). Lines 11–13 show that a candidate is added to the length-1 pattern set F_1 if its support is no less than the threshold σ. Each pattern in F_1 is represented as a triple $(s, s.count, Y)$, in which signature Y is default value. Finally, the function returns a full length-1 patten set.

Function 1. init-gen(D, σ)

Input: a sequence database D
$F_1 \leftarrow \emptyset$; // initialize F_1 to store the frequent 1-sequences
$P_1 \leftarrow \emptyset$ // initialize P_1 to store the checked subsequences
1: **for** each sequence $S \in D$ **do**
2: **for** each 1-subsequence $s \in S$ **do**
3: **if** $s \in P_1$ **then**
4: continue;

(*continued on next page*)

```
5:      else
6:        for each sequence S' ∈ (D − (S_1, S_2, ..., S_{S.id})) do
7:          if s ⊑ S' then
8:            s.count++;                    // increment the
  support count
9:          end if
10:       end for
11:       if s.count/n ⩾ σ then
12:         F_1 ← ∪_1(s, s.count, Y);       // n is |D|, Y is default
13:       end if
14:       P_1 ← ∪_1 s;
15:     end if
16:   end for
17: end for
18: Return F_1;
```

Function ConSP-gen() is invoked for mining the length-k ($k \geqslant 2$) contiguous sequential patterns. Line 1, similar to the line 3 of Function 1, shows the termination condition: when the newly split snippet has been generated in previous splitting process, such snippet is eliminated immediately. Each distinct snippet is checked by pre-post-subsequence pruning scheme, which ensures the infrequent candidates are identified and pruned before support pruning (line 3). The star character "$*$" appearing in triple $(s_{pre}, *, *)$ and $(s_{post}, *, *)$ is regarded as wildcard to accept any value. The final output is one of three groups: (1) Null; (2) P_k; and (3) P_k and F_k. The first group says that the candidate is a repeated snippet, and it is first eliminated (line 2) safely. The second one shows that at least one pre-pro-subsequence of the candidate is infrequent (line 16), or the support is less than the support threshold (line 13). The candidate is also pruned if one of the conditions is met. The last one denotes the candidate is frequent and is added to set F_k (line 11). The checked snippet set P_k is updated by adding an element if there appears a new snippet accordingly (lines 11, 13 and 16).

Function 2. ConSP-gen($D, s, F_{k-1}, P_k, S.id, \sigma$)

```
Input: a sequence database D, a candidate subsequence s, a
  set F_{k−1} with length-(k − 1) ConSPs, a set P_k with scanned
  length-k contiguous subsequences, an identifier S.id of
  sequence S in D, and a support threshold σ
1: if s ∈ P_k then
2:   Return Null                  // prune the checked snippet
3: else if (s_pre, *, *) ∈ F_{k−1} and (s_post, *, *) ∈ F_{k−1} then
4:   for each sequence S' ∈ (D − (S_1, S_2, ..., S_{S.id})) do
5:     if s ⊑ S' then
6:       s.count++;          // increment the support count
7:     end if
8:   end for
9:   if s.count/n ⩾ σ then
10:    F_k ← ∪_k(s, s.count, Y);
11:    Return F_k and P_k ← ∪_k s;
12:  else
13:    Return P_k ← ∪_k s;
14:  end if
15: else
16:   Return P_k ← ∪_k s;
17: end if
```

Function CloConSP-gen() is performed for discovering all the length-$(k-1)$ closed contiguous sequential patterns according to the set of already mined length-$(k-1)$ contiguous sequential patterns and the set of newly found length-k ones. Each discovered pattern s, as one of the attributes in tuple $(s, s.count, B)$ existing in F_k, first produces two maximal sub-patterns, i.e., pre-subsequence s_{pre} and post-subsequence s_{post}, according to Eqs. (1) and (2) (line 1). And then the function scans F_{k-1} to find such two tuples whose the first attribute values are equal to s_{pre} or s_{post} (lines 2 and 5). Tuple $(s_{pre}, s_{pre}.count, Y)$ in F_k is replaced by $(s_{pre}, s_{pre}.count, N)$ if $s_{pre}.count$ from F_{k-1} and $s.count$ from F_k are equal as well as s_{pre} and s (line 3). Similarly, tuple $(s_{post}, s_{post}.count, Y)$ is checked subsequently (line 6). By continually replacing the non-closed contiguous sequential patterns in length-$(k-1)$ pattern set, the output set F_{k-1} consists of closed contiguous sequential patterns labeled with Y and non-closed ones labeled with N.

Function 3. CloConSP-gen(F_{k-1}, F_k)

```
Input: a set F_{k−1} with length-(k − 1) ConSPs, a set F_k with
  length-k ConSPs
1: for each element (s, s.count, Y) ∈ F_k do
2:   if (s_pre, s_pre.count, Y) ∈ F_{k−1} then
3:     replace (s_pre, s_pre.count, Y) with (s_pre, s_pre.count, N) in
  F_{k−1};
4:   end if
5:   if (s_post, s_post.count, Y) ∈ F_{k−1} then
6:     replace (s_post, s_post.count, Y) with (s_post, s_post.count, N) in
  F_{k−1};
7:   end if
8: end for
   Return F_{k−1};
```

3.6. Illustrating case

Let us examine how to use CCSpan for discovering closed contiguous sequential patterns, as exemplified below. For brevity, we use Pru_1, Pru_2, and Pru_3 instead of checked snippet pruning, pre-post-subsequence pruning, and support pruning respectively.

For the same sequence database D in Table 3 with support $\sigma = 2$ shared with Example 1. The closed contiguous sequential patterns in D can be mined by CCSpan in the following steps:

1) $F_1 : (C, 4, Y), (A, 4, Y), (B, 4, Y)$ // generate 1-patterns
2) Candidate generating and pruning for 2-patterns
   s_1 $(CA, 3, Y)$, ~~(AA,1,null)~~$[Pru_3]$, $(AB, 4, Y), (BC, 4, Y)$
   s_2 ~~(AB,null,null)~~$[Pru_1]$, ~~(BC,null,null)~~$[Pru_1]$, ~~(CB,1,null)~~$[Pru_3]$
   s_3 ~~(CA,null,null)~~$[Pru_1]$, ~~(AB,null,null)~~$[Pru_1]$, ~~(BC, null,null)~~$[Pru_1]$
3) $F_2 : (CA, 3, Y), (AB, 4, Y), (BC, 4, Y)$
4) $F_1 : (C, 4, N), (A, 4, N), (B, 4, N)$ // closure checking
5) Candidate generating and pruning for 3-patterns
   s_1 ~~(CAA,null,null)~~$[Pru_2]$, ~~(AAB, null,null)~~$[Pru_2]$, $(ABC, 3, Y)$
   s_2 ~~(ABC,null,null)~~$[Pru_1]$, ~~(BCB,null,null)~~$[Pru_2]$
   s_3 ~~(CAB,1,null)~~$[Pru_3]$, ~~(ABC,null,null)~~$[Pru_1]$
6) $F_3 : (ABC, 3, Y)$
7) $F_2 : (CA, 3, Y), (AB, 4, Y), (BC, 4, Y)$ // closure checking
8) Candidate generating and pruning for 4-patterns
   s_1 ~~(CAAB,null,null)~~$[Pru_2]$, ~~(AABC, null,null)~~$[Pru_2]$
   s_2 ~~(ABCB,null,null)~~$[Pru_2]$
   s_3 ~~(CABC, null,null)~~$[Pru_2]$
9) $F_4 : NULL$
10) $F : (CA, 3, Y), (AB, 4, Y), (BC, 4, Y), (ABC, 3, Y)$

The invalid candidates are labeled with strickout and the specific pruning steps are attached inside the square brackets "[ ]". Note that the algorithm does not need to check the last sequence whether there exist distinct patterns, because all the new subsequences whose actual supports are 1, cannot satisfy the support requirement

Table 3
The characteristics of datasets.

Dataset	Seq. count	Distinct items	Avg. len.	Max. len.	DS len.
Gazelle	59,601	497	2.51	267	149,638
Mushroom	8416	119	23.00	23	193,568
Gazelle_test	10,000	454	2.49	149	24,900
Mushroom_test	10,000	119	23.00	23	230,000
Protein_10	10,000	20	17.51	20	175,137
Protein_20	20,000	20	17.49	20	349,898
Protein_30	30,000	20	17.50	20	525,040
Protein_40	40,000	20	17.50	20	699,908
Protein_50	50,000	20	17.49	20	874,581

($\sigma = 2$). We can see that most unpromising candidates are pruned by checked snippet pruning and pre-post-subsequence pruning, which contributes to the mining efficiency.

3.7. Complexity analysis

Considering the CCSpan algorithm attentively, the running time is mainly affected during snippet splitting and checked snippet pruning because most invalid candidates are pruned in such two stages. The whole time consumption is approximately the sum of splitting and checked snippet pruning cost under careful scrutiny. Without loss of generality, let k be the maximal length of patterns and S be a sequence in the original database D. The database length n is computed by $\sum_{i=0}^{|D|-1} l(S_i)$, where i is the identifier of S in D. Thus the aggregate execution times of above two procedures is $2(n + (n-1) + \cdots + (n-k+1)) = 2kn + k(k-1)$ mathematically, where k is a small constant in real-life mining task. Consequently, the complexity of CCSpan algorithm is $O(n)$, which is linearly scalable in terms of the database size.

We have carefully examined the design of CCSpan, which formulates several termination conditions to make the recursive process *return* early on, so as to accelerate the mining process of closed contiguous sequential patterns. During the mining process, we do not need to load the full data into memory. Instead, only one sequence resides in memory at any time, which decomposes the cost of allocating and freeing memory and makes mean shift well suited for discovering closed contiguous sequential patterns in big data sets.

4. Experiments

We will exhibit the thorough experimental results to verify the following claims: (1) The discovered closed contiguous sequential patterns are more compact and meanwhile have a better classification performance than the closed sequential patterns. (2) CCSpan consumes much less memory and can be an order of magnitude faster than existing algorithms when the support is low or the database is dense. (3) CCSpan holds a good scalability in both the runtime and memory usage against the increasing number of sequences for the datasets.

4.1. Datasets and environment

In our experiments, we used a variety of real and synthetic datasets to study the performance of the CCSpan algorithm. Furthermore, for thorough accessing the algorithm, we select the datasets which can cover a wide range of distribution characteristics, i.e., sparse and dense.

The first dataset, namely Gazelle, is sparse. It came from an e-commerce and has been used in KDDCup-2000 competition and is now available through the SPMF website [32]. Gazelle has three different versions: BMS1, BMS2 and BMSPos. To make our experiments fair to all the algorithms, one of our real-life datasets shares the BMS1 version with the performance study in [11]. This dataset consists of 59,601 sequences of click-stream data with an average length of 2.51 items and contains 497 distinct items.

The second dataset, Mushroom, is dense. It contains characteristics of various species of mushrooms and was originally obtained from the UCI repository of machine learning databases. This dataset consists of 119 unique items, 8416 sequences, resulting in 193,568 items. This dataset is also available at the same webside shared with the dataset Gazelle.

The third and fourth datasets, Gazelle_test and Mushroom_test, are randomly sampled from datasets Gazelle and Mushroom respectively for evaluating the quality of mined closed contiguous sequential patterns.

The fifth to ninth datasets are five synthetic datasets. They are generated via a data generator provided by [33] based on a long protein sequence WP_044990988, which is online at NCBI [34]. It is a Rhodococcus equi protein sequence consisting of 8934 amino acids, and has been annotated on many different RefSeq genomes from the same or different species. The characteristics of these datasets are shown in Table 3. More detailed information about such datasets can be referred from [32–34].

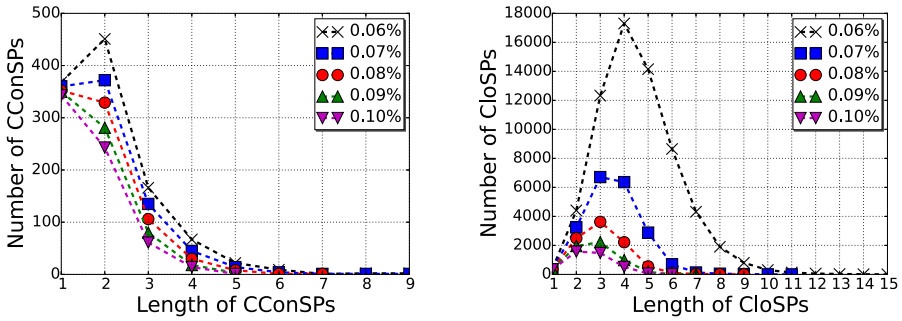
We conducted a comprehensive performance study to evaluate various aspects of algorithm CCSpan. All the experiments were conducted on a computer with Intel Core i7 2.4Ghz CPU, 8 GB memory, and Windows 7 installed. In the experiments we compared CCSpan with two classic and two recent closed sequential pattern mining algorithms, CloSpan [10], BIDE [11], ClaSP [12] and CM-ClaSP [13]. The source codes of the four closed sequential mining algorithms can be found from the SPMF data mining library [35].

4.2. Conciseness study

The effectiveness study on the two real datasets is reported as follows: Fig. 1 depicts the distribution comparison between discovered closed sequential patterns (CloSPs) and closed contiguous sequential patterns (CloConSPs) in the sparse dataset Gazelle. Fig. 1(a) shows the distribution of CloConSPs against their length for support thresholds varying from 0.06% to 0.10%, while Fig. 1(b) shows the distribution of CloSPs. From Fig. 1(a) and (b), we can see that the longest CloConSPs have a length of 9 while the longest CloSPs are up to 15 at support 0.06%. The biggest subset consisting of single-length CloConSPs occurs at length 2 while the biggest subset consisting of single-length CloSPs is at length 4. The former contains only 451 patterns but the latter more than 17,000 patterns, which is far more numerous than the former. The CloConSPs mainly appear at below length 4 while the CloSPs are from 2 to 8.

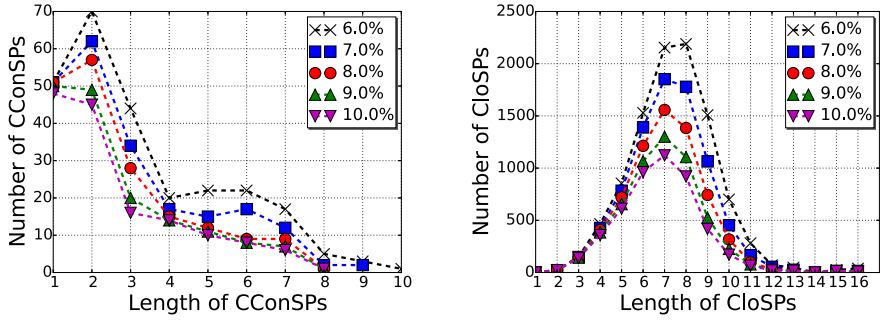
We also use the dense dataset Mushroom, to compare discovered CloConSPs with CloSPs. Fig. 2(a) shows the distribution of CloConSPs with varied support thresholds, from which one can see that all patterns are relatively short no greater than 10, while the length of CloSPs as shown in Fig. 2(a) reaches length 16. Like the trend in Fig. 1(b), when the support threshold tends towards a low value, for example, at support 6.0%, the number of CloSPs increases dramatically. Most CloConSPs fall in the spectrum of 2-patterns to 3-patterns while CloSPs at the range of 6-patterns to 9-patterns, which is longer than the former.

To further quest the performance, the full closed sequential patterns and contiguous sequential patterns are compared as shown in Fig. 3 in terms of the total number of patterns. From Fig. 3(a), the numbers of closed sequential patterns formed by previous algorithms vary from 3974 ($\sigma = 0.10\%$) to 64,762 ($\sigma = 0.06\%$), while that of closed contiguous sequential patterns range from



(a) Distribution of CloConSPs (b) Distribution of CloSPs

Fig. 1. Distribution comparison between CloConSPs and CloSPs on Gazelle.



(a) Distribution of CloConSPs (b) Distribution of CloSPs

Fig. 2. Distribution comparison between CloConSPs and CloSPs on Mushroom.

662 to 1088, which are much less than the former with the corresponding supports. Furthermore, in Fig. 3(b) the set of CloConSPs is much more compact than that of CloSPs. When the supports of Fig. 3(a) and (b) are respectively lowered to 0.06% and 6.0%, the total numbers of closed sequential patterns grows up to 64.7 K and 10 K, each of which is much more than the size of the original databases. While the closed contiguous sequential pattern mining algorithm CCSpan, only generates 1088 and 255 patterns respectively with the same threshold.

The average lengths of CloConSPs and CloSPs are also presented for varied support thresholds. From Fig. 4(a), the average length of identified CloConSPs increases from 1.62 to 2.06, while that of CloSPs from 2.59 to 4.58. Of the two patterns, the length of the former is almost half of that of the latter. Similarly, on dataset Mushroom as shown in Fig. 4(b), the average length of CloConSPs ranges from 2.64 to 3.3, while that of CloSPs from 6.79 to 7.42. For both dataset Gazelle and dataset Mushroom, the average length of the closed contiguous sequential patterns is much shorter than that of closed sequential patterns with the same support threshold.

Although the characteristics of the two datasets are quite different, the above experiments deliver the following observations: (1) The CloConSPs are concentrated in the spectrum of shorter length compared with the distribution of CloSPs. (2) The size of CloConSPs' set is far less than that of CloSPs' under a common support threshold. (3) The set of discovered CloConSPs maintains shorter patterns than that of CloSPs with the same support threshold.

4.3. Quality evaluation

One typical application of frequent sequential patterns is to build classifiers by using these patterns as features. In the sequel, we will evaluate the power of the closed contiguous sequential patterns in terms of both the recall and the classification efficiency.

Two datasets, Gazelle_test and Mushroom_test, are generated by randomly sampling from Gazelle and Mushroom (see Table 3). Each dataset consists of 10 K sequences. Tables 4 and 5 show that the closed contiguous sequential patterns hold the equivalent recall with varied support thresholds in both sparse and dense datasets. (The evaluation classifiers are available online at [36].)

Fig. 5 depicts the runtime of classification of the two patterns (CloConSPs and CloSPs) in both sparse and dense datasets. Fig. 5(a) illustrates the processing time of the classifiers at different support thresholds in dataset Gazelle_test. The classifier with CloConSPs is about ten times at $\sigma = 0.10\%$ and at least 110 times at $\sigma = 0.06\%$ faster than the classifier with CloSPs with the same support value. In dense dataset Mushroom_test, the CloConSP's classifier is more than 40 times faster than the CloSP's one for any minimum value.

There are three major reasons for the boosted performance of CloConSPs for classification. First, the number of CloConSPs is much less than that of CloSPs, but the CloConSPs carry the same information as well as the latter. Second, the average length of CloConSPs is shorter than that of CloSPs, thus reducing the runtime of classification. Third, the CloConSPs maintain contiguous constraint, which are preferred over the CloSPs for pattern matching.

Considering Tables 4 and 5, different patterns CloConSPs and CloSPs have the same expressive power partly because the 1-patterns of them match substantial numbers of examples on Gazelle_test and Mushroom_test. In feature selection for classification, length-1 patterns are often discarded because they fetch a lot of negative examples as well as positive ones. To further evaluate the classification performance of CloConSPs, we remove the 1-patterns from the the sets of CloConSPs and CloSPs. Moveover, we randomly generate two sequence sets as negative example sets using the items of datasets Gazelle and Mushroom. The sequences

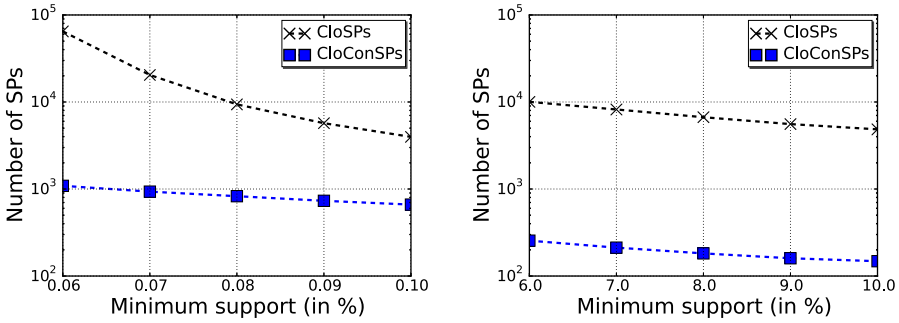


Fig. 3. Number comparison between CloConSPs and CloSPs.

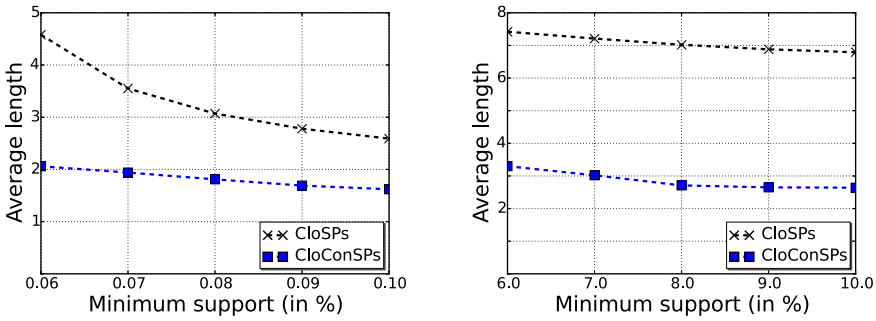


Fig. 4. Average length comparison between CloConSPs and CloSPs.

Table 4
Recall comparison between CloConSPs and CloSPs on Gazelle_test.

Pattern	0.06%	0.07%	0.08%	0.09%	0.10%
CloSPs	0.9961	0.9958	0.9951	0.9951	0.9941
CloConSPs	0.9961	0.9958	0.9951	0.9951	0.9941

(as positive examples) selected randomly from the real datasets along with the negative ones form two testing datasets Gazelle_PN and Mushroom_PN, each of which consists of 10,000 positive sequences and 10,000 negative ones. Tables 6 and 7 show the classification comparisons between CloConSPs and CloSPs in terms of the three evaluation metrics *Recall*, *Precision*, and F_1-*score*. From Table 6, one can see that the set of CloConSPs has a lower recall but a higher precision and a competitive F_1-score on sparse dataset Gazelle_PN compared with the one of CloSPs. Table 7 shows that the set of CloConSPs obtains a equivalent recall, a higher precision, and a better F_1-score on dense dataset Mushroom_PN. Fig. 6 shows the F_1-*score* comparison between CloConSPs and CloSPs for classification on both sparse and dense datasets, from which one can see that the CloConSPs achieve a higher F_1-*score* with varied support thresholds in comparison with the CloSPs.

The above performances deliver the effectiveness of CCSpan. First, the set of closed contiguous sequential patterns generated by CCSpan is over an order of magnitude less size than the set of closed sequential pattern by CloSpan, BIDE, ClaSP and CM-ClaSP in all databases, especially when the support threshold is low or the database is rich in frequent patterns. Second, the mined closed contiguous sequential patterns share the same information with the closed sequential patterns. Third, the set of closed contiguous sequential patterns holds a steadily better performance than the set of closed sequential patterns for sequence classification.

Table 5
Recall comparison between CloConSPs and CloSPs on Mushroom_test.

Pattern	6.0%	7.0%	8.0%	9.0%	10.0%
CloSPs	1.0000	1.0000	1.0000	1.0000	1.0000
CloConSPs	1.0000	1.0000	1.0000	1.0000	1.0000

4.4. Efficiency study

We first test CCSpans efficiency for both sparse and dense datasets in terms of the execution time. Fig. 7(a) presents the running time of the five algorithms at different support thresholds in dataset Gazelle. At a high support, CCSpan can be several times slower than the other four algorithms, but once we lower the support to a certain point, for example, at $\sigma = 0.06\%$, three algorithms including CloSpan, ClaSP and CM-ClaSP discontinue searching patterns because they run out of memory. For such case, CCSpan becomes more efficient than BIDE, especially when the support is lowered to 0.05%, it can be over an order of magnitude faster than BIDE. Fig. 7(b) shows the runtime efficiency comparison for the dataset Mushroom among CCSpan, CloSpan and BIDE rather than all five algorithms, because ClaSP and CM-ClaSP algorithms ran out of memory with any support threshold on such dataset. The experimental result makes clear distinction among the algorithms tested. It shows the same ordering of the algorithms for runtime: "*CCSpan* < *CloSpan* < *BIDE*". For any minimum support, CCSpan is about one and more than two orders of magnitude faster than CloSpan and BIDE respectively. Overall, CCSpan significantly outperforms the previous closed sequential pattern mining algorithms when the support is low or the dataset is dense regarding runtime efficiency.

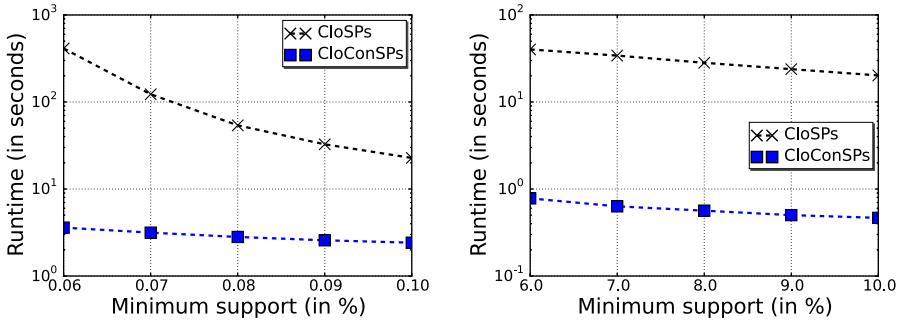


Fig. 5. Runtime comparison between CloConSPs and CloSPs for classification.

Table 6
Classification comparison between CloSPs and CloConSPs on Gazelle_PN.

	Pattern	0.06%	0.07%	0.08%	0.09%	0.10%
Recall	CloSPs	0.9761	0.9661	0.9523	0.9395	0.9300
	CloConSPs	0.8872	0.8684	0.8537	0.8309	0.8073
Precision	CloSPs	0.6877	0.7254	0.7586	0.7868	0.8112
	CloConSPs	0.9768	0.9805	0.9821	0.9837	0.9857
F_1-*score*	CloSPs	0.8069	0.8286	0.8445	0.8564	0.8665
	CloConSPs	0.9298	0.9211	0.9134	0.9009	0.8876

We next compare the memory consumption among the five algorithms for both dataset Gazelle and dataset Mushroom. Fig. 8(a) shows the results for dataset Gazelle, from which one can see that CCSpan is efficient in memory usage. It requires a much smaller memory space in comparison with the other four algorithms. In most cases, CloSpan, ClaSP and CM-ClaSP algorithms outperform BIDE in memory usage. However, when $\sigma = 0.06\%$, such three algorithms terminate due to out of memory. Like Fig. 7(b), three algorithms, i.e., CCSpan, CloSpan and BIDE, are compared as shown in Fig. 8(b) regarding space usage since the other two algorithms (ClaSP and CM-ClaSP) ran out of memory with any support threshold. All the three algorithms consume relatively stable memory space, while CCSpan occupies over an order of magnitude less memory than both CloSpan and BIDE. From the above two sub-figures, it can be demonstrated that CCSpan significantly outperforms the other four algorithms in memory usage in all cases. Our memory usage analysis also shows part of the reason why some algorithms become really slow because the huge number of patterns may consume a tremendous amount of memory.

From the above efficiency study, we can conclude that CCSpan has good overall performance for both dense and sparse datasets among the five algorithms tested.

4.5. Scalability study

In order to test the scalability of CCSpan, we randomly sampled a series of datasets from a long protein sequence WP_044990988 (for details, see Table 3), which are very dense. As we explained earlier, both ClaSP and CM-ClaSP fail to finish the searching due to out of memory. Thus Figs. 9 and 10 show the scalability outcomes of just three algorithms (CCSpan, CloSpan and BIDE) regarding runtime and memory usage. Fig. 9 shows the running time of the above algorithms with database size ranging from 10 K to 50 K and with the support as 1.0%. We see that CCSpan is slightly faster than CloSpan, while it uses over one order of magnitude less runtime than BIDE.

Fig. 10 shows the curves of space usage of CCSpan at $\sigma = 1.0\%$, with the database size growing from 10 K to 50 K too. We can see that CCSpan is efficient in memory usage. It consumes over an order of magnitude less memory than both CloSpan and BIDE. For example, on dataset Protein_30K, CloSpan occupies 1077 MB memory and BIDE 906 MB memory, while CCSpan only uses 64 MB memory. It is easy to see that CCSpan holds a more memory efficient for a wide range of database size compared with the other two algorithms.

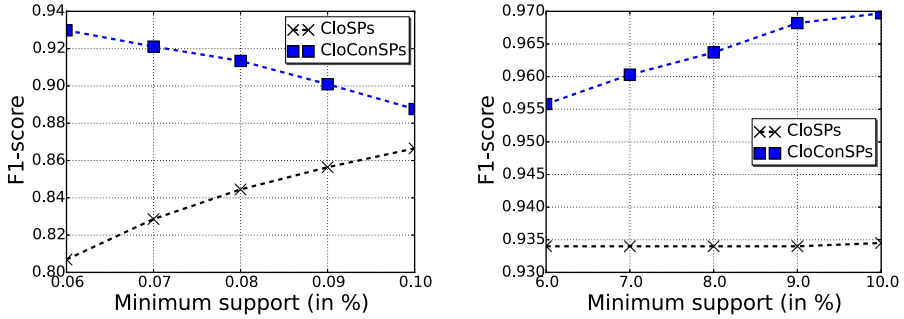
The experimental outputs demonstrate the scalability of CCSpan from both runtime and memory usage. That is to say, it has very good scalability in terms of the database size.

Table 7
Classification comparison between CloConSPs and CloSPs on Mushroom_PN.

	Pattern	6.0%	7.0%	8.0%	9.0%	10.0%
Recall	CloSPs	1.0000	1.0000	1.0000	1.0000	1.0000
	CloConSPs	1.0000	1.0000	1.0000	1.0000	1.0000
Precision	CloSPs	0.8761	0.8761	0.8761	0.8761	0.8771
	CloConSPs	0.9153	0.9237	0.9300	0.9383	0.9412
F_1-*score*	CloSPs	0.9340	0.9340	0.9340	0.9340	0.9345
	CloConSPs	0.9558	0.9603	0.9637	0.9682	0.9697

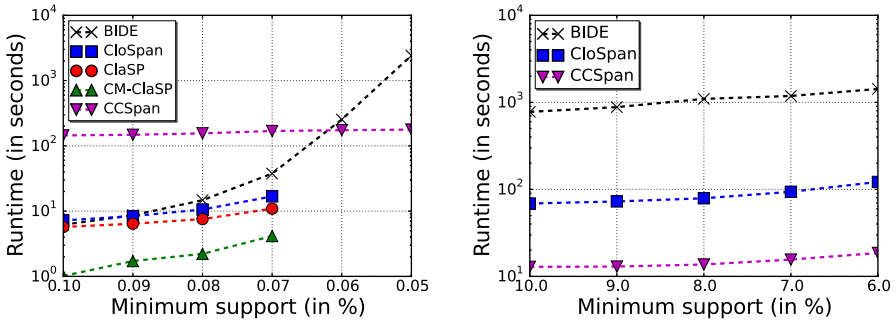
5. Related work

Sequential patterns mining has been investigated for years. GSP [37], SPADE [38] and PrefixSpan [8] are three typical algorithms proposed for mining the complete set of general sequential patterns, where PrefixSpan has the best overall performance [39]. Yun et al. [16–20] proposed a weighted sequential mining algorithm with the aim of pushing the weight constraint into the mining while maintaining the downward closure property. Four typical algorithms, i.e., CloSpan [10], BIDE [11], ClaSP [12] and CM-ClaSP [13] were introduced in a row for performing the closed sequential pattern mining. CloSpan is based on the search framework of PrefixSpan and also uses projected databases to recursively mine closed sequential patterns. BIDE uses a sequence closure checking scheme called BI-Directional Extension and prunes the search space by BackScan pruning strategy. CloSpan and BIDE often terminate their operations since the daunting number of candidate sub-patterns or frequent sub-trees causes out of memory or intractable workload. ClaSP and CM-ClaSP algorithms use a vertical data representation and some pruning schemes to mine the closed sequential patterns, which have a better performance in terms of runtime compared with CloSpan and BIDE. However, when using a very low support threshold or a very dense database, they discontinue their running due to out of memory. While we adopt a new snippet-growth paradigm to generate potential frequent patterns which accurately record the items occurrences in the original



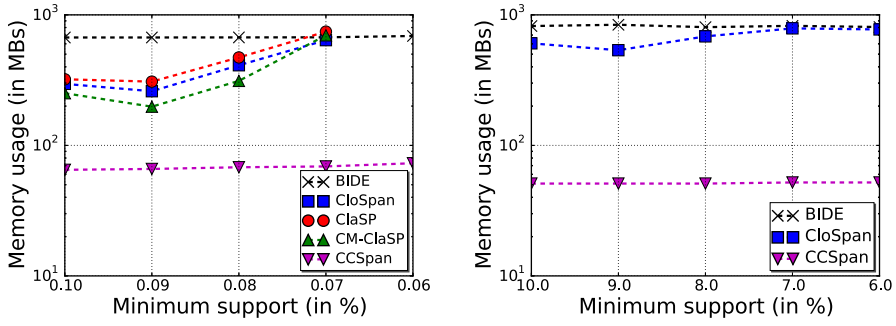
(a) F_1-*score* on Gazelle_PN (b) F_1-*score* on Mushroom_PN

Fig. 6. F_1-*score* comparison between CloConSPs and CloSPs for classification.



(a) Runtime on dataset Gazelle (b) Runtime on dataset Mushroom

Fig. 7. Runtime comparison of the five algorithms on two datasets.



(a) Memory usage on Gazelle (b) Memory usage on Mushroom

Fig. 8. Memory usage comparison of the five algorithms on two datasets.

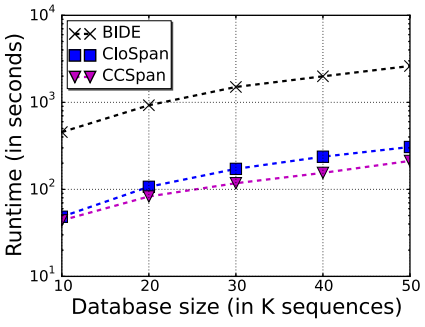


Fig. 9. Runtime on Protein series.

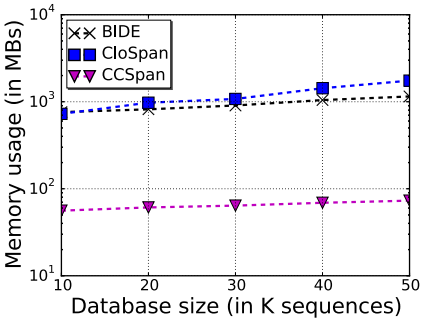


Fig. 10. Memory usage on Protein series.

sequences, and launch three pruning techniques to prune the futile parts of search space.

CloSpan, BIDE, ClaSP and CM-ClaSP focus on the sequential pattern mining with closed constraint only. However, we combine the closed and the contiguous constraints for closed contiguous sequential pattern mining. There exist several important differences. First, the closed contiguous sequential patterns achieve a competitive performance in some particular applications, especially in biological sequence analysis, which is very different from closed sequential patterns generated by previous algorithms. Second, the closed contiguous sequential patterns have a smaller volume and a shorter average length yet the same informative closed sequential patterns, which are preferred for building classifiers in comparison with the closed sequential patterns. Third, our thorough performance studies with both sparse and dense datasets demonstrated that the pruning techniques are effective in pruning the unpromising parts of search space, and CCSpan holds a better performance in terms of both efficiency and scalability compared with previous algorithms.

6. Conclusion

We investigated the combination of the contiguous and the closed constraints for a compact and lossless sequential pattern mining, to address the closed sequential pattern mining problem, that could be spawning a daunting number of inefficient and redundant patterns. We presented a novel algorithm, CCSpan, which efficiently mines closed contiguous sequential patterns. CCSpan adopts a new snippet-growth paradigm to generate potential frequent patterns which accurately record the items' occurrences in the original sequences. Meanwhile, CCSpan launches three pruning techniques to prune the futile parts of search space. A complete set of closed contiguous sequential patterns is generated by performing the closure checking.

We evaluated the performance of CCSpan using several real-life datasets with varied densities. Experimental results demonstrated that the set of closed contiguous sequential patterns discovered by CCSpan is much more compact than the set of closed by the state-of-the-art algorithms (CloSpan, BIDE, ClaSP and CM-ClaSP), especially when feeding a low support threshold or a pattern-enriched database. We also demonstrated the closed contiguous sequential patterns generated by CCSpan carry the same information and have a better classification performance in comparison with closed sequential patterns. Moreover, CCSpan is efficient and scalable in terms of database size.

There are many interesting issues related to CCSpan that can be investigated further, such as mining maximal contiguous sequential patterns, mining top-k closed contiguous sequential patterns, and extension of the method toward classifying sequences for particular applications are interesting issues for future research.

Acknowledgements

The authors are grateful to Assistant Prof. P. Fournier-viger for releasing the source codes of the compared algorithms. Also, they would like to express their thanks to Prof. T. Li, Prof. J. Cao, Dr. J. Guo, Dr. L. Tao, and Dr. C. Zhang for helpful discussions and suggestions during the development of CCSpan and helpful comments on the manuscript. In addition, they would also like to thank the anonymous reviewers and editors. This work was supported by the National Natural Science Foundation of China (NSFC) under the Grant No. 61375053.

References

[1] E.-C. Lu, W.-C. Lee, V.S. Tseng, A framework for personal mobile commerce pattern mining and prediction, IEEE Trans. Knowl. Data Eng. 24 (5) (2012) 769–782.
[2] P. Fournier-Viger, U. Faghihi, R. Nkambou, E.M. Nguifo, Cmrules: mining sequential rules common to several sequences, Knowl.-Based Syst. 25 (1) (2012) 63–76.
[3] D. Fradkin, F. Mörchen, Mining sequential patterns for classification, Knowl. Inform. Syst. (2015) 1–19.
[4] P. Senkul, S. Salin, Improving pattern quality in web usage mining by using semantic information, Knowl. Inform. Syst. 30 (3) (2012) 527–541.
[5] W. Qu, Y. Jia, M. Jiang, Pattern mining of cloned codes in software systems, Inform. Sci. 259 (2014) 544–554.
[6] J.H. Chang, Mining weighted sequential patterns in a sequence database with a time-interval weight, Knowl.-Based Syst. 24 (1) (2011) 1–9.
[7] V.C.-C. Liao, M.-S. Chen, Dfsp: a depth-first spelling algorithm for sequential pattern mining of biological sequences, Knowl. Inform. Syst. 38 (3) (2014) 623–639.
[8] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu, Mining sequential patterns by pattern-growth: the prefixspan approach, IEEE Trans. Knowl. Data Eng. 16 (11) (2004) 1424–1440.
[9] K. Kaneiwa, Y. Kudo, A sequential pattern mining algorithm using rough set theory, Int. J. Approx. Reason. 52 (6) (2011) 881–893.
[10] X. Yan, J. Han, R. Afshar, Clospan: mining closed sequential patterns in large datasets, in: Proceedings of SIAM International Conference on Data Mining, SIAM, 2003, pp. 166–177.
[11] J. Wang, J. Han, Bide: efficient mining of frequent closed sequences, in: Proceeding of 20th International Conference on Data Engineering, IEEE, 2004, pp. 79–90.
[12] A. Gomariz, M. Campos, R. Marín, B. Goethals, Clasp: an efficient algorithm for mining frequent closed sequences, in: Advances in Knowledge Discovery and Data Mining, Springer, 2013, pp. 50–61.
[13] P. Fournier-Viger, A. Gomariz, M. Campos, R. Thomas, Fast vertical mining of sequential patterns using co-occurrence information, in: Advances in Knowledge Discovery and Data Mining, Springer, 2014, pp. 40–52.
[14] C. Luo, S.M. Chung, Efficient mining of maximal sequential patterns using multiple samples, in: SDM, SIAM, 2005, pp. 415–426.
[15] P. Fournier-Viger, C.-W. Wu, V.S. Tseng, Mining maximal sequential patterns without candidate maintenance, in: Advanced Data Mining and Applications, Springer, 2013, pp. 169–180.
[16] G. Lee, U. Yun, K.H. Ryu, Sliding window based weighted maximal frequent pattern mining over data streams, Expert Syst. Appl. 41 (2) (2014) 694–708.
[17] U. Yun, H. Shin, K.H. Ryu, E. Yoon, An efficient mining algorithm for maximal weighted frequent patterns in transactional databases, Knowl.-Based Syst. 33 (2012) 53–64.
[18] U. Yun, G. Lee, K.H. Ryu, Mining maximal frequent patterns by considering weight conditions over data streams, Knowl.-Based Syst. 55 (2014) 49–65.
[19] U. Yun, G. Pyun, E. Yoon, Efficient mining of robust closed weighted sequential patterns without information loss, Int. J. Artif. Intell. Tools 24 (01) (2015) 1550007.
[20] U. Yun, E. Yoon, An efficient approach for mining weighted approximate closed frequent patterns considering noise constraints, Int. J. Uncertain., Fuzz. Knowl.-Based Syst. 22 (06) (2014) 879–912.
[21] B. Le, M.-T. Tran, B. Vo, Mining frequent closed inter-sequence patterns efficiently using dynamic bit vectors, Appl. Intell. (2015) 1–11.
[22] M. Fabrègue, A. Braud, S. Bringay, F. Le Ber, M. Teisseire, Mining closed partially ordered patterns, a new optimized algorithm, Knowl.-Based Syst. 79 (2015) 68–79.
[23] N. Hariri, B. Mobasher, R. Burke, Context-aware music recommendation based on latenttopic sequential patterns, in: Proceedings of the Sixth ACM Conference on Recommender Systems, ACM, 2012, pp. 131–138.
[24] J. Chen, T. Cook, Mining contiguous sequential patterns from web logs, in: Proceedings of the 16th International Conference on World Wide Web, ACM, 2007, pp. 1177–1178.
[25] J. Chen, Contiguous item sequential pattern mining using updown tree, Intell. Data Anal. 12 (1) (2008) 25–49.
[26] T.H. Kang, J.S. Yoo, H.Y. Kim, Mining frequent contiguous sequence patterns in biological sequences, in: Bioinformatics and Bioengineering, 2007. BIBE 2007. Proceedings of the 7th IEEE International Conference on, IEEE, 2007, pp. 723–728.
[27] M. Karim, M. Rashid, B.-S. Jeong, H.-J. Choi, et al., An efficient approach to mining maximal contiguous frequent patterns from large dna sequence databases, Genom. Inform. 10 (1) (2012) 51–57.
[28] J. Rissanen, Minimum description length principle, Encyclopedia Mach. Learn. (2010) 666–668.
[29] Z. Zeng, J. Wang, J. Zhang, L. Zhou, Fogger: an algorithm for graph generator discovery, in: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, ACM, 2009, pp. 517–528.
[30] J. Wang, J. Han, C. Li, Frequent closed sequence mining without candidate maintenance, IEEE Trans. Knowl. Data Eng. 19 (8) (2007) 1042–1056.

[31] J. Zhang, Y. Wang, D. Yang, Automatic learning common definitional patterns from multi-domain wikipedia pages, in: 2014 IEEE International Conference on Data Mining Workshop (ICDMW), IEEE, 2014, pp. 251–258.
[32] Spmf, 2015. <http://www.philippe-fournier-viger.com/spmf/index.php>.
[33] Dataset Generator, 2015. <http://cit.sjtu.edu.cn/DatasetGenBio.aspx>.
[34] Ncbi, 2015. <http://www.ncbi.nlm.nih.gov>.
[35] P.F. Viger, A. Gomariz, T. Gueniche, A. Soltani, C.-W. Wu, V.S. Tseng, Spmf: a java open-source pattern mining library, J. Mach. Learn. Res. 15 (2014) 3389–3393.
[36] Recall, 2015. <http://cit.sjtu.edu.cn/ClassifierCon.aspx>.
[37] R. Srikant, R. Agrawal, Mining Sequential Patterns: Generalizations and Performance Improvements, Springer, 1996.
[38] M.J. Zaki, Spade: an efficient algorithm for mining frequent sequences, Mach. Learn. 42 (1–2) (2001) 31–60.
[39] J. Han, H. Cheng, D. Xin, X. Yan, Frequent pattern mining: current status and future directions, Data Min. Knowl. Disc. 15 (1) (2007) 55–86.