Mining Contiguous Sequential Generators in Biological Sequences

Jingsong Zhang, Yinglin Wang, Chao Zhang, and Yongyong Shi

Abstract—The discovery of conserved sequential patterns in biological sequences is essential to unveiling common shared functions. Mining sequential generators as well as mining closed sequential patterns can contribute to a more concise result set than mining all sequential patterns, especially in the analysis of big data in bioinformatics. Previous studies have also presented convincing arguments that the generator is preferable to the closed pattern in inductive inference and classification. However, classic sequential generator mining algorithms, due to the lack of consideration on the contiguous constraint along with the lower-closed one, still pose a great challenge at spawning a large number of inefficient and redundant patterns, which is too huge for effective usage. Driven by some extensive applications of patterns with contiguous feature, we propose ConSgen, an efficient algorithm for discovering contiguous sequential generators. It adopts the n-gram model, called shingles, to generate potential frequent subsequences and leverages several pruning techniques to prune the unpromising parts of search space. And then, the contiguous sequential generators are identified by using the equivalence class-based lower-closure checking scheme. Our experiments on both DNA and protein data sets demonstrate the compactness, efficiency, and scalability of ConSgen.

Index Terms—Sequential pattern mining, closed sequential pattern, sequential generator, contiguous sequential generator, DNA sequence, protein sequence, motif finding

1 Introduction

Conserved frequent subsequences (motifs) in biological sequences often reflect functionally meaningful shared features. Sequential pattern mining, which identifies frequent subsequences as patterns in sequence databases, is an important data mining issue in bioinformatics communities. It has shown board applications, including motif finding and analysis [1], [2], [3], [4], [5], DNA sequence analysis [6], [7], [8], protein sequence analysis [9], [10], and antimicrobial develop [11]. The mining approaches have been studied extensively, including general sequential pattern mining [12], closed sequential pattern (CSP) mining [13], [14], sequential generator mining [15], [16], [17], maximal sequential pattern mining [18], [19], and interesting sequential pattern mining [20], [21], [22].

Due to the equivalence class principle [23], the problem of mining compact representations of sequential patterns has attracted more attention recently. Closed sequential patterns and sequential generators are two classic patterns with compact yet lossless compression. The former are upper-closed while the latter lower-closed. Within an equivalence class, the average length of the generators is no more than that of closed patterns. Thus, the generator is preferable to the closed pattern in inductive inference and classification.

Some algorithms have been developed for mining sequential generators [15], [16], [17], such mining often produces a large number of frequent patterns satisfying the support threshold, especially when the support is low or the database is rich in frequent patterns. For example, biology domain, the set of sequential generators derived by current mining methods, has a significantly greater size than the corresponding database and even the total length of the database. Such result set is too huge for effective usage, which is a rather ticklish problem facing previous pattern mining algorithms. In addition, for generating such massive patterns, the mining process is prohibitively expensive naturally. Classic sequential generator mining algorithms, including A priori-based and pattern-growth, share a poor scalability in terms of support threshold and database density because too many frequent generators will occupy much memory and incur large search space for closure checking of new patterns or pattern growth, which usually occurs when the low support threshold or the dense database is used. The huge number of sequential generators also makes some further data analysis like feature selection and classification a prohibitive task. Hence, what sequential patterns need to be mined remains a problem, which must be more compact and complete compared to the sequential generators.

With the increasing availability of biological sequences, the mining of sequential generators with a certain specific constraint has attracted much more attention since it can lead to more concise patterns. One example is mining the contiguous sequential generators, in which the items appearing in the sequences that contain the pattern must be

- *J. Zhang is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: jasun@dmbio.info.*
- *Y. Wang is with the Department of Computer Science and Technology, Shanghai University of Finance and Economics, Shanghai 200433, China. E-mail: wang.yinglin@shufe.edu.cn.*
- *C. Zhang is with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801. E-mail: czhang82@illinois.edu.*
- *Y. Shi is with the Bio-X Institutes, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: shiyongyong@gmail.com.*

Manuscript received 20 Apr. 2015; revised 24 Sept. 2015; accepted 9 Oct. 2015. Date of publication 26 Oct. 2015; date of current version 4 Oct. 2016. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TCBB.2015.2495132

adjacent w.r.t. the underlying ordering as typically defined in the sequential generator [17]. Using sequential patterns with contiguous feature can greatly benefit a wide spectrum of real-life tasks as described before. Additionally, based on a common sequence database, the average length of the contiguous sequential generators is much shorter than that of the 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 principle (MDL) [24], [25], which has a sound statistical foundation rooted in the well-known Bayesian inference and Kolmogorov complexity.

Existing techniques developed for mining compact sequential patterns have focused on the study of scalable methods for general sequential generator mining, which does not involve the contiguous feature of patterns. Such techniques cannot be directly applied to contiguous sequential generator (ConSG) mining. This is because the set of contiguous sequential generators is not a proper subset of the set of sequential generators or general sequential patterns. Existing methods using a post-pruning step do not perform the mining task. Moreover, such methods do not record the pattern's occurrences in the database. By pushing the contiguous constraint (i.e., pre-pruning) into the mining process, they will cost more memory space and running time. Consequently, the major challenge is how to design an effective algorithm to ensure the result patterns are contiguous and meanwhile lower-closed. In the sequels, we will call the sequential patterns satisfying the contiguous yet lower-closed constraints *contiguous sequential generator*, and seek to mine them in an effective and efficient fashion.

In this paper, we propose ConSgen, which employs three steps to effectively identify contiguous sequential generators. In the first step, ConSgen splits each sequence of database into a series of snippets using the n-gram model. The items in each snippet strictly keep the adjacency and ordering of the original sequences. The generation of such snippets avoids enumerating all possible joints of frequent items, but guarantee information equivalency. In the second step, ConSgen first uses a repeated snippet checking method to ensure any unique snippet is detected only once. Next, it checks the snippets' frequency by a technique called max-prefix-suffix pruning, and counts the surviving snippets to determine if they are frequent. Such pruning techniques greatly reduce the search space of contiguous sequential generators. In the third step, ConSgen introduces an equivalence class-based lower-closure checking scenario to efficiently find the contiguous sequential generators.

Our contributions are summarized as follows:

- We introduce the problem of mining contiguous sequential generators in biological sequences. To the best of our knowledge, we are first in attempting to discover contiguous sequential generators that reflect the adjacency of items and the lower-closure of patterns simultaneously.
- We develop ConSgen for the proposed problem. ConSgen algorithm does not rely on enumerating all possible combinations of frequent subsequences to produce potential longer patterns. Such algorithm and the related datasets are available on our public website, http://www.dmakd.com/ConSgen.aspx.
- Our extensive experiments on both DNA and protein data sets show that, ConSgen is scalable to discover contiguous sequential generators against support threshold, and it outperforms compared methods significantly in terms of compactness and efficiency.

The structure of this paper is as follows. In Section 2, we focus on the formulation of contiguous sequential generator mining problem as well as some notations used throughout the paper. Section 3 studies some properties of contiguous sequential generators and draws a solution to discover the contiguous sequential generators. Experimental results are reported in Section 4, followed by the related work in Section 5. Section 6 concludes the paper.

2 Preliminaries and Problem Statement

In this section, we first introduce some preliminary concepts, and then formalize and exemplify the problem of contiguous sequential generator mining, which provide a necessary background for our algorithm development.

2.1 Preliminary Concepts

An *item* is a basic unit and can be a base or an amino acid in bioinformatics. Let $I = \{i_1, i_2, \ldots, i_m\}$ be a non-empty finite set consisting of distinct items. A *sequence* $S = < a_1, a_2, \ldots, a_n >$ is an ordered list, which can be a series of base pairs or amino acids accordingly. For brevity, a sequence is also expressed as $s = a_1 a_2 \ldots a_n$. According to the definitions, notice that an item a_i ($i \in [1, n]$) can occur more than once in a sequence s. The *size* of a sequence, denoted as $|S|$, is the number of unique items in the sequence. The *length* of a sequence, denoted as $l(S)$, is the total number of items in the sequence, i.e., $l(S) = n$. A sequence with k items is also termed a **length-k sequence** or **k-sequence**[1] for short. The number of distinct items in such a k-sequence is less than or equal to k. For example, one sequence $CAGTTCGCGC$ is a 10-sequence while its size is 4. With these basic notations, we further define some notations and terms to smooth the analyzing of ConSgen.

By examining the definitions of both closed itemset [26] and closed sequential pattern [27], they share a common feature that is the longest priority. We call such longest priority feature *upper-closure* or say the closed itemsets and the closed sequential patterns are *upper-closed*. Similarly, We call the shortest priority feature *lower-closure* or say the itemset generators and the sequential generators are *lower-closed*.

Definition 1. *Given two sequences $S_1 = < a_1 a_2 \cdots a_i >$ and $S_2 = < b_1 b_2 \cdots b_j >$, 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 $k_1, k_2, \cdots, k_i$ such that: (1) $1 \leq k_1 < k_2 < \cdots < k_i \leq j$; 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*

1. In this paper, k-subsequence and k-pattern are used and indicate the similar meanings.

TABLE 1
An Example Sequence Database D

Sequence Id	Sequence
1	CAAGC
2	AGCGT
3	CAGC
4	AGGCA

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.,* $Sup^r_D(s) = |\{S|S \in D \wedge s \sqsubseteq S\}|/|D|$.

For the convenience of description, we use support $Sup_D(s)$ instead of both absolute support $Sup^a_D(s)$ and relative support $Sup^r_D(s)$ if not explicitly stated in the rest of the pater.

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

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

Obviously, contiguous sequential generators satisfy the lower-closed constraint shared with itemset and sequential generators.

Definition 5. *Given two sequences $s_1 = < a_1a_2 \cdots a_i >$ and $s_2 = < b_1b_2 \cdots b_j >$, 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) $Sup_D(s_1) = Sup_D(s_2)$.*

Definition 6. *Given two sequences $s_1 = < a_1a_2 \cdots a_i >$ and $s_2 = < b_1b_2 \cdots b_j >$, s_1 is a* ***max-prefix*** *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) $a_1 = b_1$, $a_2 = b_2, \ldots, a_i = b_{j-1}$.*

Definition 7. *Given two sequences $s_1 = < a_1a_2 \cdots a_i >$ and $s_2 = < b_1b_2 \cdots b_j >$, s_1 is a* max-suffix *of s_2, denoted as $s_1 \sqsubset_{suf} s_2$, if (1) $l(s_1) \geqslant 1$, $l(s_2) - l(s_1) = 1$; and (2) $a_1 = b_2$, $a_2 = b_3, \ldots, a_i = b_j$.*

For simplicity, if the context is clear we appellatively call max-prefix and max-suffix *max-prefix-suffix* (or *prefix-suffix* for short) in the remainder of the paper.

2.2 Problem Statement

Let's first use a example shown in the following to convey the differences of four sequential patterns, i.e., sequential pattern (SP), closed sequential pattern, sequential generator (SG), and contiguous sequential generator.

Example 1. Given a sequence database D as shown in Table 1 and an absolute support $\sigma = 2$. The input database exhibited in the second column of the table has a total of four distinct items (base pairs) and four input sequences (i.e., $|D| = 4$). Four types of frequent patters with their absolute supports attached after ":", are elicited from D respectively (see Table 2). The full set of frequent sequences contains 17 patterns, excluding the empty pattern ϕ, which is trivially frequent.[2] While the full set of closed sequential patterns is only 6. Similarly, the whole set of sequential generators consists of 11 patterns. In contrast, the size of contiguous sequential generator set is only 5. The third column shows the average lengths of the above four patterns. It is easy to see that the average length of sequential generators is smaller than that of both sequential patterns and closed sequential patterns, while the average length of the contiguous sequential generators is even smaller than that of sequential generators. Informally, given an input sequence database and a user-specified support threshold, the size of contiguous sequential generator set is much smaller than that of sequential generator set. Moreover, the contiguous sequential generators hold the smallest average length compared to three other patterns.

TABLE 2
Comparison of Four Types of Frequent Patterns

Sequence type	Frequent patterns	Avg. length
SP	A:4, AA:2, AG:4, AGG:2, AGC:4, AC:4, G:4, GG:2, GC:4, C:4, CA:3, CAG:2, CAGC:2, CAC:2, CG:3, CGC:2, CC:2	2.235
CloSP	AA:2, AGG:2, AGC:4, CA:3, CAGC:2, CG:3	2.667
SG	A:4, C:4, G:4, AA:2, CC:2, CG:3, GG:2, CA:3, CAG:2, CAC:2, CGC:2	2.000
ConSG	C:4, A:4, G:4, CA:3, AGC:3	1.600

Problem statement. *Given a sequence database D and a minimum support σ, the problem of mining contiguous sequential generators is to discover the full set of contiguous subsequences with lower-closure feature.*

A full set of sequential patterns and that of contiguous sequential patterns can be partitioned into some equivalence classes. For a pattern s, a classification function can be defined as $f(s) = \{S \in D|s \sqsubseteq S\}$ [23], where S is a sequence in database D. Assume EC is one of the equivalence classes of a full set F. If there exist two patterns $s_1 \in EC$ and $s_2 \in EC$, then $f(s_1) = f(s_2)$. Specifically, given a contiguous frequent sequential pattern set F, we define a function for the equivalence class in F: $EC_F = \{s \in F|\forall(s_i, s_j) \wedge (Sup_D(s_i) = Sup_D(s_j)) \wedge \exists(s' \in F) \wedge (s_i \sqsubseteq s' \wedge s_j \sqsubseteq s')\}$. Note that such function is independent of the database D, which is preferred for the closure checking. Consider the definition of contiguous sequential pattern as shown in Definition 3. The pattern set F consists of a series of directed acyclic graphs. These graphs can be transformed to some full binary trees, each node of which corresponds to a frequent pattern. According to the function EC_F, each complete equivalence class falls in one of the full binary trees and shares the same actual support. Mathematically, the problem of mining contiguous sequential generators can be converted to the identifying and partitioning of equivalence classes in F. Given an equivalence class EC, the full set of

2. For brevity, we ignore the empty pattern without affecting the demonstration in the rest of this paper.

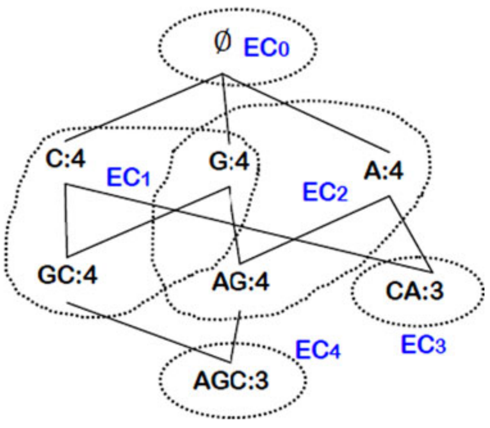


Fig. 1. Contiguous sequential patterns and equivalence classes.

generators in EC is $\{s \in EC | (\nexists s')(s' \in EC) \wedge (s \neq s' \wedge s' \sqsubseteq s)\}$ or $\{s \in EC | (\nexists s')(s' \in EC) \wedge (s' \sqsubset s)\}$.

Fig. 1 shows all the contiguous sequential patterns of the database in Table 1. There are five equivalence classes marked by dotted lines, i.e., EC_0 to EC_4, where EC_0 is an empty one. Generally, a non-empty equivalence class contains one or more levels. When there is only one level, all the patterns of it are both generators and closed patterns. For example, equivalence classes EC_3 and EC_4, each of them only has one level. Thus patterns $<CA>$ and $<AGC>$ are both contiguous generators and closed contiguous patterns. When the number of levels is two or more, the patterns at the bottom are the closed patterns while those at the top are the generators. Therefore, the sets of contiguous generators of EC_1 and EC_2 are $\{<C>, <G>\}$ and $\{<G>, <A>\}$ respectively. By finding such patterns at the top of each equivalence class in F, we can obtain the full generators as shown in the last row of Table 2.

3 Development of Efficient Mining Method

In this section, we first perform step-by-step analysis to develop an efficient method for mining contiguous sequential generators, then present the ConSgen algorithm in detail.

The algorithm works on a three-step manner. In the first step, 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 [22] in subsequent scans. Next step, by exploring some properties of contiguous sequential generators, three effective pruning techniques, redundant snippet pruning, max-prefix-suffix pruning, and support pruning are proposed 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. Such a process iterates until no more distinct patterns exist. A full set consisting of all contiguous sequential patterns is obtained accordingly. In the third step, all contiguous sequential non-generators are distinguished successively from the set of contiguous sequential patterns by performing the lower-closure checking, as suggested in Definition 4 and Theorem 2. In summary, By splitting the input sequences, progressively pruning them via three techniques and then using the the lower-closure constraint, ConSgen can discover contiguous sequential generators effectively.

3.1 Candidate Generation by N-Gram Model

Many conventional algorithms developed for mining sequential generators, due to ignoring some specific features of sequential patterns, need to enumerate all possible combinations of frequent subsequences to produce potential longer patterns, resulting in a heavy burden of both memory and time usage. In addition, two adjacent elements (or items) in a pattern (see F_{SP}, F_{CloSP}, and F_{SG} as shown in Table 2) 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 use the n-gram model 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 combinations of frequent subsequences, which is inherently costly in both runtime and space usage.

Let S be a sequence in sequence database 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 one single item. And these snippets are checked if they are frequent. In the second stage, named 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. Let us examine how to use the n-gram model for splitting the input sequences based on our running example.

Example 2. The first sequence of Table 1, i.e., $S_1 = <CAAGC>$, is reused in this example. In C_1-splitting stage, a 1-subsequence set, i.e., $C_1 = \{C, A, A, G, C\}$[3] is formed by splitting the sequence S_1. And then the C_2-splitting generates a 2-subsequence set $C_2 = \{CA, AA, AG, GC\}$ that is contains four elements (subsequences). The remaining sets produced by ConSgen are $C_3 = \{CAA, AAG, AGC\}$, $C_4 = \{CAAG, AAGC\}$, and $C_5 = \{CAAGC\}$ respectively.

The splitting manner seems rather brute force at first glance. In practice, the process terminates its splitting far before the snippets actually reach length-5 by employing several wise pruning paradigms, which will be elaborated in next section.

3.2 Search Space Pruning

By using the n-gram model detailed in Example 2, every sequence of database is discretized into a series of snippets as candidate subsequences which are rather coarse because

3. The repeated elements such1 as items A and C, will be first discarded in next pruning step.

they may contain a large number of inefficient and redundant patterns. To substantially reduce the memory usage and search space, we examine some properties of contiguous sequential patterns in the sequel, which underpin the design of pruning schemes.

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 A priori [28]. 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 max-prefix s_{pre} and max-suffix s_{suf} 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 max-prefix $s_{pre} \in F$ and the max-suffix $s_{suf} \in F$. Then both 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 max-prefix or max-suffix of s, i.e., $Sup_D(s_{pre}) \ngeqslant \sigma$ or $Sup_D(s_{suf}) \ngeqslant \sigma$ holds, then s can be safely pruned.*

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

The pruning process of ConSgen gives itself the minimal burden to run, representing a three-section manner toward contiguous sequential generator mining.

Redundant snippet pruning. In real-world datasets, some snippets often appear multiple times, not only among different sequences but also 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 redundant snippet pruning is assigned in the first step to ensure the repeated snippets are pruned as early as possible, which avoids unnecessary max-prefix-suffix 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 largely speed up ConSgen's mining process.

Max-prefix-suffix pruning. Unlike classic candidate *enumerate-and-test* paradigm, max-prefix-suffix 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 the max-prefix-suffixes of s. The length-k snippet is pruned immediately if its max-prefix or max-suffix is infrequent as indicated by Lemma 2. Obviously, in the worst case, most previous techniques need to check $C_k^{k-1} = k$ times. In contrast, our pruning scheme takes at most two times. Thus, such a 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 the pruning cost is nearly negligible compared to the aggregate running time.

Support pruning. A snippet is called a promising candidate if it meets: (1) it is distinct from the previous snippets (already split snippets); and (2) both the max-prefix and the max-suffix of the snippet are frequent. Such snippets satisfying the above two conditions can be shifted to count their supports and check whether they are frequent. For each promising candidate s, it is natural to use the conventional matching method to count the actual support, however, ConSgen does not need to check whether every sequence in the database contains s. Instead, it only checks the sequences from the next one of the current sequence identifier to the end of the database, which effectively prevents redundant snippet matching operations and accelerates the counting process.

A set consisting of length-k contiguous sequential patterns is formed when no more new k-patterns appear by iterating over above process, and the complete set of contiguous sequential k-generators can be discovered by performing the sequel *pattern lower-closure checking* technique.

3.3 Equivalence Class-Based Lower-Closure Checking

Next, ConSgen will distinguish contiguous sequential non-generators from the full set of contiguous sequential patterns. The problem is to check for each pattern s, whether there is a super-pattern absorbing s, according to the definition of equivalence class and Definition 5. Obviously a naive method, which compares each pattern with other patterns in the discovered set, is infeasible due to its $O(N^2)$ complexity. An interesting finding is that the contiguous sequential generators hold the transitivity among pattern subsets, each of which consists of single-length frequent patterns. Such a property is launched by ConSgen for pattern lower-closure checking to achieve excellent 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. □

Assume 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 is lower-closed if there exists no element of F_{k-1} which is absorbed by s with the same support. Specifically, each pair of max-prefix-suffixes of the length-k contiguous sequential patterns are first calculated by Equations (1) and (2) of [14]. 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 max-prefix-suffix and the support count of current k-pattern based on Definition 4. Depending on the definition of equivalence class within contiguous sequential pattern set, a length-k contiguous sequential pattern is lower-closed if the above two-fold conditions are satisfied simultaneously. This length-k checking is continued until every element of the length-k 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 lower-closure checking step can be conducted on-the-fly. By progressively checking, the whole contiguous sequential non-generators are efficiently identified and the remaining are generated using the equivalence class theory and the lower-closure property.

3.4 ConSgen Algorithm

In this section, for elaborating ConSgen, we first introduce several data structures. Three concise data structures are employed for performing our mining task: 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 contiguous sequential generator and non-generator 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 lower-closure, while N the non-lower-closure. The final output of the algorithm is a set F, which consists of all contiguous sequential patterns including generators and non-generators. The inside patterns of F can be organized into a set $\{\{F_1\}, \{F_2\}, \ldots, \{F_k\}\}$ consisting of k different partitions, each of which is a subset of single-length patterns.

Algorithm 1. ConSgen(D, σ)

```
Input: sequence database $D$, support threshold $\sigma$
    $F \leftarrow \emptyset$;    // initialize $F$ to store the ConSGs
    $F_k \leftarrow \emptyset$;   // initialize $F_k$ to store the length-$k$ ConSPs
 1: $F_{k-1} \leftarrow$ init-gen($D, \sigma$) // generate the frequent 1-ConSPs
 2: $F_k \leftarrow \cup_{k-1} F_{k-1}$;
 3: for each $F_{k-1} \neq \emptyset$ do
 4:   $F_k \leftarrow$ ConSP-gen($D, F_{k-1}, \sigma$);
 5:   if $F_k \neq \emptyset$ then
 6:     $F_k \leftarrow$ ConSG-gen($F_{k-1}, F_k$);
 7:     $F \leftarrow \cup_k F_k$;
 8:   end if
 9:   $F_{k-1} \leftarrow F_k$;
10: end for
Output: $F$;
```

Algorithm 1 sketches ConSgen algorithm that mines the set of contiguous sequential generators. As shown, given a transformed database D and a support threshold σ, we define global variables F and F_k to store all length contiguous sequential generators and only length-k contiguous sequential patterns respectively. The frequent sequential 1-pattern set F_1 is first derived by running the init-gen() function (subroutine, line 1). Such 1-patterns are directly added to set F without checking the lower-closure property since length-1 patterns are all generators based on Definition 4 (line 2). Those 1-patterns are treated as $(k-1)$-generators to feed ConSP-gen() function for checking longer contiguous sequential patterns (2-patterns). Based on the database D, support threshold σ and the $(k-1)$-patterns just generated, function ConSP-gen() produces a set of length-specified contiguous sequential patterns (i.e., k-patterns). Subsequently, according to 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, the length-k contiguous contiguous generators in F_k are distinguished from general contiguous sequential patterns by calling ConSG-gen() effectively (line 6). ConSP-gen() and ConSG-gen() are performed alternately until the output set F_k is empty.

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

Function init-gen(), as the first yet unique pass, is run for finding all the frequent length-1 sequential patterns, which are fed to the identification of 2-patterns. Compared with the function ConSP-gen(), init-gen() does not need to check the frequency of the max-prefix-suffixes since each candidate pattern only has one item. Such two functions share the rest of the procedure, which will be detailed in next illustration of function ConSP-gen().

Function 1. init-gen(D, σ)

```
Input: a sequence database $D$
    $F_1 \leftarrow \emptyset$; // initialize $F_1$ to store the frequent 1-patterns
    $P_1 \leftarrow \emptyset$; // 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;
 5:     else
 6:       for each sequence $S' \in (D - (S_1, S_2, \ldots, S_{S.id}))$ do
 7:         if $s \sqsubseteq S'$ then
 8:           $s.count++$; // increment the support count
 9:         end if
10:       end for
11:       if $s.count/n \geq \sigma$ then
12:         $F_1 \leftarrow \cup_1 (s, s.count, Y)$; // $n$ is $|D|$, $Y$ is default
13:       end if
14:       $P_1 \leftarrow \cup_1 s$;
15:     end if
16:   end for
17: end for
    Return $F_1$;
```

Function ConSP-gen() is invoked for discovering the length-k ($k \geqslant 2$) contiguous sequential patterns. Those candidate k-subsequences provided by splitting the initial sequences are eliminated safely if they exist in the snippet set P_k, because all elements of it have been checked before (line 3 and 4). Each distinct snippet is checked by max-prefix-suffix pruning scheme, which ensures the infrequent candidates are identified and pruned before support pruning (line 5). The star character "$*$" appearing in triple

$(s_{pre}, *, *)$ and $(s_{suf}, *, *)$ is regarded as wildcard to accept any value. 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-k pattern set F_k if its support is no less than the threshold σ. Each pattern in F_k is represented as a triple $(s, s.count, Y)$, in which signature Y is default value. Finally, the function returns a full length-k contiguous sequential pattern set.

Function ConSG-gen() is performed for identifying all the length-k contiguous sequential generators 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 subsequences, i.e., max-prefix s_{pre} and max-suffix s_{suf}, according to Equations (1) and (2) of [14] (line 1). And then the function scans F_{k-1} to find such two tuples whose first attribute values are equal to s_{pre} or s_{suf} (line 2 and 5). Tuple $(s, s.count, Y)$ in F_k is replaced by $(s, s.count, N)$ 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 $(s, s.count, Y)$ is checked subsequently (line 6). By continually replacing the contiguous sequential non-generators in length-k pattern set, the output set F_k consists of contiguous sequential generators labeled with Y and non-generators labeled with N.

Function 2. ConSP-gen(D, F_{k-1}, σ)

```
Input: a sequence database $D$, support threshold $\sigma$,
   $(k-1)$-pattern set $F_{k-1}$
   $F_k \leftarrow \emptyset$; // initialize $F_k$ to store the frequent $k$-patterns
   $P_k \leftarrow \emptyset$; // initialize $P_k$ to store the checked subsequences
 1: for each sequence $S \in D$ do
 2:   for each $k$-subsequence $s \in S$ do
 3:     if $s \in P_k$ then
 4:       continue;
 5:     else if $(s_{pre}, *, *) \in F_{k-1}$ and $(s_{suf}, *, *) \in F_{k-1}$ then
 6:       for each sequence $S' \in (D - (S_1, S_2, \ldots, S_{S.id}))$ do
 7:         if $s \sqsubseteq S'$ then
 8:           $s.count++$; // increment the support count
 9:         end if
10:       end for
11:       if $s.count/n \geq \sigma$ then
12:         $F_k \leftarrow \cup_k(s, s.count, Y)$; // $n$ is $|D|$, $Y$ is default
13:       end if
14:     end if
15:     $P_k \leftarrow \cup_k s$;
16:   end for
17: end for
   Return $F_k$;
```

Considering the ConSgen algorithm attentively, the running time is mainly affected during snippet splitting and redundant snippet pruning because most invalid candidates are pruned in such two stages. The whole time consumption is approximately the sum of splitting and redundant snippet pruning cost under careful scrutiny. Without loss of generality, let k be the maximal length of patterns and S be a sequence in 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. And then the aggregate execution times of above two procedures is $2kn + k(k-1)$ mathematically, where k is a small constant in real-life mining task. Consequently, the complexity of ConSgen algorithm is $O(n)$, which is linearly scalable in terms of the database size.

Function 3. ConSG-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 $(s, s.count, Y) \in F_k$ do
2:   if $(s_{pre}, s_{pre}.count, Y) \in F_{k-1}$ then
3:     replace $(s, s.count, Y)$ with $(s, s.count, N)$ in $F_k$;
4:   end if
5:   if $(s_{suf}, s_{suf}.count, Y) \in F_{k-1}$ then
6:     replace $(s, s.count, Y)$ with $(s, s.count, N)$ in $F_k$;
7:   end if
8: end for
   Return $F_k$;
```

We have carefully examined the design of ConSgen, which formulates several termination conditions to make the recursive process *return* early on, so as to accelerate the identifying process of contiguous sequential generators. 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 eliminates the cost of allocating and freeing memory and makes mean shift well suited for discovering contiguous sequential patterns in big data sets.

4 Empirical Result

In this section, we will report the a series of experimental results to verify the following claims: (1) The set of discovered contiguous sequential generators is more compact than that of general sequential generators. (2) ConSgen holds the significantly better efficiency with varied support thresholds compared to the state-of-the-art algorithms. (3) ConSgen has a good scalability for the biological sequence databases in terms of both memory and runtime usage.

4.1 Test Environment and Data Sets

We performed an extensive performance study to evaluate various aspects of algorithm ConSgen. All the experiments were conducted on a computer with Intel Core i7 2.4 Ghz CPU, 8 GB memory, and Windows 7 installed. In the experiments we compared ConSgen with a sequential pattern mining algorithm PrefixSpan [12], two closed sequential pattern mining algorithms CloSpan [27] and BIDE [29], and three sequential generator mining algorithms VGEN [15], FEAT [16] and FSGP [17]. The source codes of preceding six algorithms can be derived from the SPMF data mining library [30].

This section presents the results of our experimental study performed on both DNA and protein datasets. The first dataset, AX829204, can be downloaded from the National Center for Biotechnology Information website. It is a Homo Sapiens (human) DNA sequence consisting of 12,680 base pairs (bp), and has been studied for the diagnosis of breast cancer. We randomly sampled 1,000 sequences with a length-20 to form the dataset for our practical

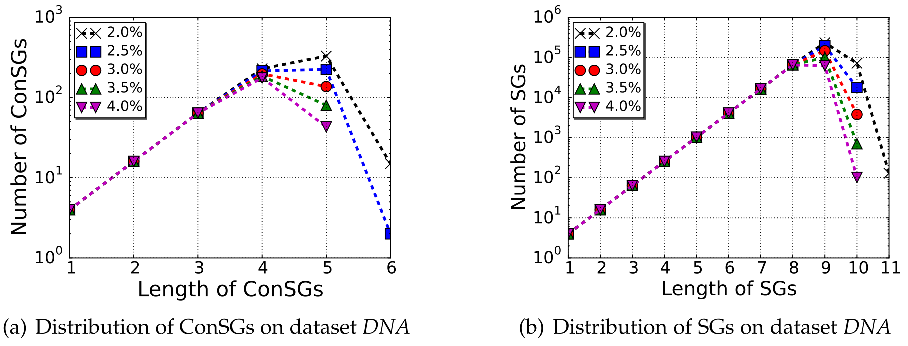


Fig. 2. Distribution comparison between the two patterns on dataset *DNA*. (a) The distribution of ConSGs against their length for support thresholds varying from 2.0 to 4.0 percent. (b) The distribution of SGs against their length for support thresholds shared with Fig. 2a.

experiments. The second dataset, WP_044990988, is also available at the above website. It is a Rhodococcus equi protein sequence consisting of 8,934 amino acids, and has been annotated on many different RefSeq genomes from the same or different species. Similarly, A dataset consisting of 1,000 20-sequences are generated for our experiments. The DNA dataset is dense, while the protein dataset is a little sparse. For brevity, we call AX829204 a dataset *DNA* and WP_044990988 a dataset *Protein* in the remainder of the paper.

4.2 Compactness Study

The compactness study in the two real datasets is reported as follows: Fig. 2 depicts the distribution comparison between discovered contiguous sequential generators (ConSGs) and sequential generators (SGs) on dataset *DNA*. Fig. 2a shows the distribution of ConSGs against their length for support thresholds varying from 2.0 to 4.0 percent, while Fig. 2b shows the distribution of SGs. From the above two sub-figures, we can see that the longest ConSGs have a length of 6 while the longest SGs are up to 11 at support 2.0 percent. The biggest subset consisting of single-length ConSGs occurs at length 5 while the biggest subset consisting of single-length CloSPs is at length 9. The 5-pattern subset in Fig. 2a contains only 323 patterns but the 9-pattern subset in Fig. 2b more than 232k ones, which is far more numerous than the former.

We also used the dataset *Protein* to compare the distribution of discovered ConSGs and SGs. Fig. 3a shows the distribution of the number of the ConSGs against the length of themselves with varied support thresholds. We can see that the most patterns are quite short, while the maximal length of SGs as shown in Fig. 3b reaches length 7. In common with Fig. 2b, when the pre-specified support threshold tends towards a low value, for example, at support 1.5 percent, the number of SGs increases dramatically.

To further quest the performance, the sets of frequent sequential patterns, closed sequential patterns and sequential generators were obtained by performing PrefixSpan, CloSpan, and VGEN algorithms in aforementioned two datasets respectively. Fig. 4 shows the distributions of four pattern groups, i.e., SPs, CloSPs, SGs, and ConSGs, each of which is displayed with varied support thresholds for the two datasets. In Fig. 4a, the numbers of sequential generators formed by previous algorithms vary from 149,136 ($\sigma = 4.0\%$) to 392,029 ($\sigma = 2.0\%$), which are more than 149 times larger than the size of initial dataset. While the numbers of contiguous sequential generators are only from 303 to 657 at the corresponding support thresholds. Fig. 4b shows the similar observation. From Figs. 4a and 4b, we can see that the set of SGs is slightly more compact than that of SPs, while the set of ConSGs is much more compact than that of SGs. In addition, for all the test cases where the datasets varied from dense to little sparse, the set of contiguous sequential generators is more concise than that of sequential generators and the performance gap gets larger and larger.

The average lengths of SGs and ConSGs are also presented for support thresholds varying from 4.0 to 2.0 percent. From Fig. 5a, the average length of identified ConSGs increases from 3.79 to 4.38, while that of SGs from 8.22 to 8.89. Of the two patterns, the length of the former is almost half of that of the latter. Similarly, on dataset *Protein*, the lengths of SGs and ConSGs have the similar performance as shown in Fig. 5b. For both dataset *DNA* and dataset *Protein*,

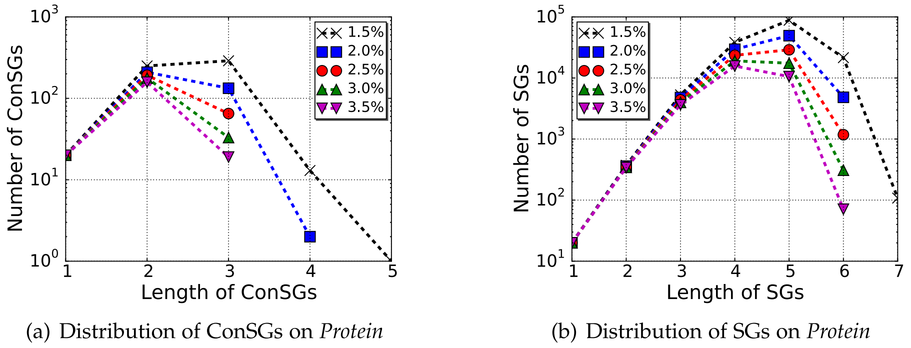
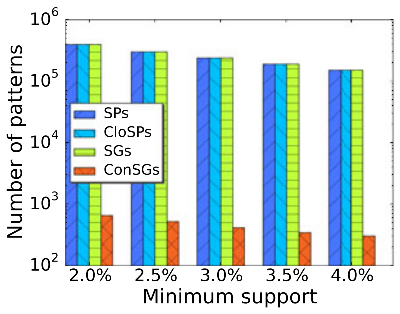
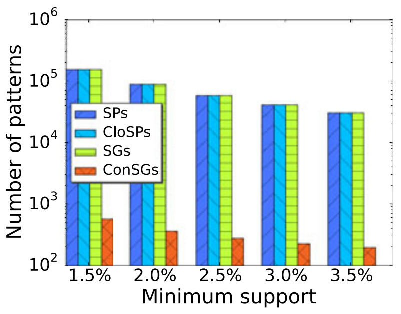


Fig. 3. Distribution comparison between the two patterns on dataset *Protein*. (a) The distribution of ConSGs against their length for support thresholds varying from 1.5 to 3.5 percent. (b) The distribution of SGs against their length for support thresholds shared with Fig. 3a.

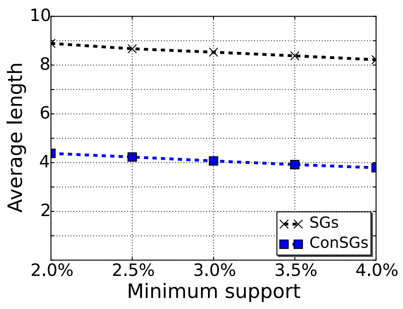


(a) Pattern number on *DNA*

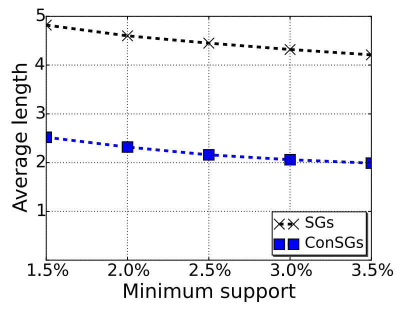


(b) Pattern number on *Protein*

Fig. 4. Pattern Number comparison of the four patters on two datasets. (a) Pattern number comparison among SPs, CloSPs, SGs, and ConSGs with σ ranging from 2.0 to 4.0 percent on dataset *DNA*. (b) Pattern number comparison among the four patterns with σ ranging from 1.5 to 3.5 percent on dataset *Protein*.



(a) Avg. len. of patterns on *DNA*



(b) Avg. len. of patterns on *Protein*

Fig. 5. Average length comparison between SGs and ConSGs. (a) Average length comparison between SGs and ConSGs with σ ranging from 2.0 to 4.0 percnt on dataset *DNA*. (b) Average length comparison of the two patterns with σ ranging from 1.5 to 3.5 percent on dataset *Protein*.

the average length of the contiguous sequential generators is much shorter than that of general sequential generators with the same support threshold.

The above performances demonstrate the compactness of ConSgen. The set of contiguous sequential generators generated by ConSgen is more than two orders of magnitude less size than the set of SPs, CloSPs, and SGs by PrefixSpan, CloSpan, and VGEN on both DNA and protein sequence datasets, especially when the support threshold is low or the dataset is rich in frequent patterns. In addition, the average length of contiguous sequential generators is shorter than that of sequential generators and thus the contiguous sequential generators are preferable in terms of sequence analysis.

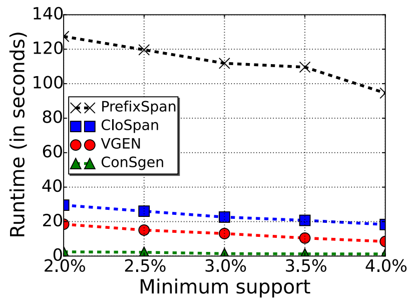
4.3 Efficiency Study

We tested ConSgens efficiency in both runtime and memory usage for the two real datasets in terms of the support threshold. Table 3 shows the processing time of the seven well-known algorithms at different support thresholds on dataset *DNA*. We can find that the three algorithms under the dotted line are dramatically slower than the first four ones. For example, FEAT's runtime grows up to 10,702.359 seconds (nearly three hours) when the support threshold is 2 percent. For presenting the fine-grained runtime of ConSgen, Fig. 6a illustrates the curves of processing time of the four faster algorithms only. The testing result makes clear distinction among the algorithms tested. It shows the same ordering of the algorithms for runtime: "$ConSgen < VGEN < CloSpan < PrefixSpan$". For any minimum support, ConSgen is at least six times faster than VGEN while VGEN is faster than CloSpan and much faster than PrefixSpan. Fig. 6b shows the runtime on dataset *Protein* and the details are shown in Table 4. We can see that VGEN is slightly faster than CloSpan and CloSpan slightly faster than PrefixSpan with the minimum support threshold ranging from a low of 2.0 percent to a high of 3.5 percent. While ConSgen is significantly faster than VGEN, CloSpan, and PrefixSpan for any support threshold. From Fig. 6 and the above two tables, we observe that ConSgen consumes much less running time in comparison with three other algorithms, especially when the support is low or the dataset is dense.

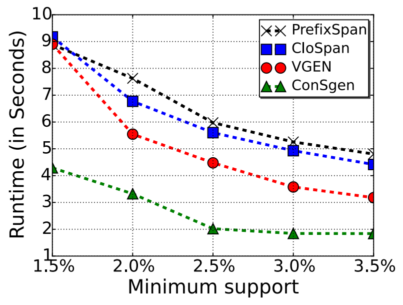
We compare the memory consumption among the four algorithms, ConSgen, VGEN, CloSpan, and PrefixSpan for both dataset *DNA* and dataset *Protein*. Fig. 7a shows the results for DNA dataset, from which we can see that ConSgen is efficient in memory usage. It consumes more than an order of magnitude less memory than VGEN, CloSpan, and PrefixSpan. For any minimum support, ConSgen only needs about 52 MB memory space while three other algorithms cost more than 320 MB. Fig. 7b exhibits the memory usage

TABLE 3
Runtime on Dataset *DNA* (in Seconds)

Alogrithm	2.0%	2.5%	3.0%	3.5%	4.0%
PrefixSpan	127.421	119.652	111.836	109.615	94.554
CloSpan	29.549	26.053	22.623	20.720	18.318
VGEN	18.493	15.073	13.067	10.455	8.479
ConSgen	2.465	2.204	1.424	1.278	1.246
BIDE	852.838	757.647	679.089	666.343	585.893
FSGP	4,531.531	2,326.169	1,496.641	1,037.078	700.397
FEAT	10,702.359	7,362.832	5,840.557	4,576.874	3,486.744



(a) Runtime on dataset *DNA*



(b) Runtime on dataset *Protein*

Fig. 6. Runtime comparison of the four algorithms on two datasets. (a) Runtime comparison among PrefixSpan, CloSpan, VGEN, and ConSgen with σ ranging from 2.0 to 4.0 percent on dataset *DNA*. (b) Runtime comparison of the four algorithms with σ ranging from 1.5 to 3.5 percent on dataset *Protein*.

for protein dataset, from which we can see that CloSpan is also more efficient than three other algorithms as well as on DNA dataset. From the above two sub-figures, it can be demonstrated that CloSpan significantly outperforms VGEN, CloSpan, and PrefixSpan 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 occupy a tremendous amount of memory.

The above measures demonstrate the efficiency of ConSgen. The capability of ConSgen in runtime remains more stable than the state-of-the-art approaches: VGEN, CloSpan, PrefixSpan, BIDE, FSGP, and FEAT over the range of support thresholds.

4.4 Scalability Study

For testing the scalability of ConSgen, we also randomly sampled four other DNA datasets and four other protein datasets from dataset AX829204 and dataset WP_044990988 respectively. Moreover, both dataset *DNA* and dataset *Protein* as sampled earlier are added and reused in the following experiments.

Fig. 8a shows the space usage scalability tests of the four algorithms on dense dataset *DNA*, with the database size growing from 1 to 5 K sequences, and with support threshold $\sigma = 3.0\%$. We can see that ConSgen is efficient in memory usage. It consumes more than an order of magnitude less memory than VGEN, CloSpan, and PrefixSpan. For example, on dataset DNA_3K, CloSpan occupies 670 MB memory and PrefixSpan 764 MB memory while ConSgen only uses 51 MB memory. The sequential generator mining algorithm, VGEN, can not complete the searching when the database size is more than 2 K because it ran out of memory. Similarly, Fig. 8b shows the space usage scalability test

TABLE 4
Runtime on Dataset *Protein* (in Seconds)

Alogrithm	1.5%	2.0%	2.5%	3.0%	3.5%
PrefixSpan	8.892	7.628	5.975	5.257	4.805
CloSpan	9.174	6.772	5.602	4.930	4.416
VGEN	8.900	5.546	4.476	3.577	3.179
ConSgen	4.290	3.324	2.021	1.844	1.839
BIDE	85.450	61.686	48.904	41.902	35.17
FSGP	592.676	172.945	71.870	33.478	18.205
FEAT	893.854	458.831	290.619	193.728	130.158

results using the *Protein* dataset series with support threshold $\sigma = 2.5\%$. On such sparse datasets, ConSgen is also more memory efficient for a wide range of database size compared to three other algorithms.

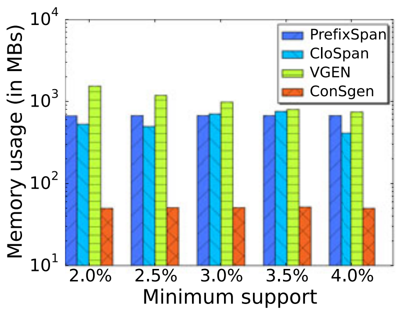
We also tested the runtime scalability of the four algorithms on dataset series of the *DNA* and *Protein* with $\sigma = 3.0\%$ and $\sigma = 2.5\%$ when database size is varied from 1 to 5 K as shown in Fig. 9. From Fig. 9a, we see that ConSgen consumes more than an order of magnitude less runtime than both VGEN and CloSpan, while it uses more than two orders of magnitude less runtime than PrefixSpan. As we noted earlier, VGEN runs out of memory when the database size grows up to 3 K. Fig. 9b shows the results of scalability tests on the sparse dataset series. The VGEN algorithm is slightly faster than CloSpan and CloSpan slightly faster than PrefixSpan. While ConSgen has the least runtime compared to three other algorithms.

The above results demonstrate the scalability of ConSgen. Specifically, it has very good scalability in terms of both the database size and the support threshold. In addition, it is more stable in both memory and time consumption than VGEN, CloSpan, and PrefixSpan.

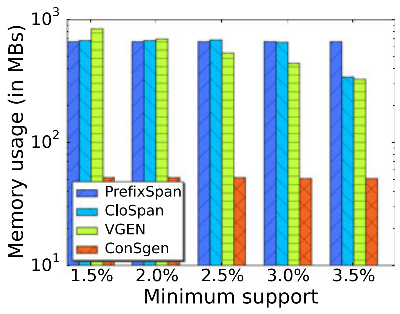
5 Related Work

Frequent sequential pattern mining, first introduced by Agrawal and Srikant [31], has been widely studied and many efficient algorithms have been proposed [12], [32], [33], [34], [35]. Three typical algorithms, namely GSP [32], SPADE [34], and PrefixSpan [12] are successively proposed for mining the full set of general sequential patterns. Overall, PrefixSpan has a better performance [36] compared to the previous sequential pattern mining algorithms. Due to the exponential number of sequential patterns, many studies have focused on developing concise representations of sequential patterns[13], [15], [16], [17], [19], [20], [21], [22], [27], [37], [38], [39]. Among these patterns, closed sequential patterns [13], [27], [37] and sequential generators [15], [16], [17] have attracted a great deal of attention since they are two important lossless compression of frequent sequential patterns.

Two famous algorithms, CloSpan and BIDE, are successively introduced by Yan et al. [27] and Wang et al. [13] for mining closed sequential patterns. The former is based on the search framework of PrefixSpan and also uses projected databases to recursively mine closed sequential patterns. The latter uses a sequence closure checking scheme called

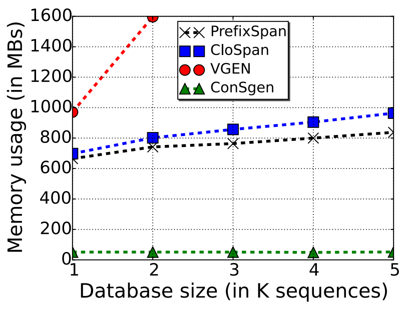


(a) Memory usage on *DNA*

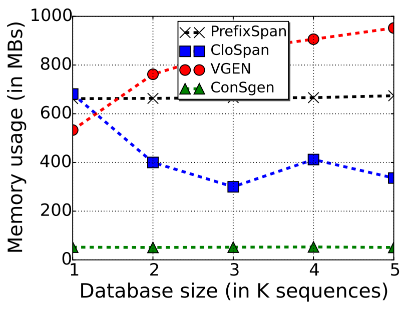


(b) Memory usage on *Protein*

Fig. 7. Memory usage comparison of four algorithms on two datasets. (a) Memory usage comparison among PrefixSpan, CloSpan, VGEN, and ConSgen with σ ranging from 2.0 to 4.0 percent on dataset *DNA*. (b) Memory usage comparison of the four algorithms with σ ranging from 1.5 to 3.5 percent on dataset *Protein*.

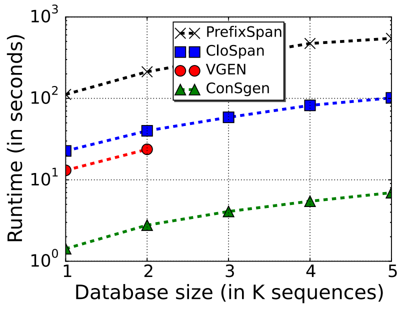


(a) Memory usage on *DNA*

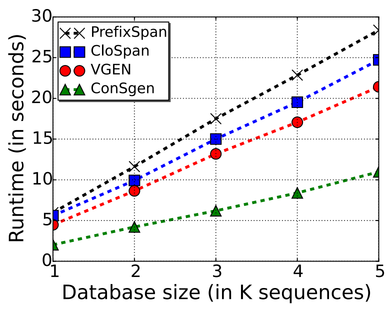


(b) Memory usage on *Protein*

Fig. 8. Scalability test of the memory usage. (a) Memory usage comparison among PrefixSpan, CloSpan, VGEN, and ConSgen for database sizes varying from 1 to 5 K sequences on dataset *DNA*. (b) Memory usage comparison among the four algorithms for database sizes varying from 1 to 5 K sequences on dataset *Protein*.



(a) Time usage on *DNA*



(b) Time usage on *Protein*

Fig. 9. Scalability test of the time usage. (a) Runtime comparison among PrefixSpan, CloSpan, VGEN, and ConSgen for database sizes varying from 1 to 5 K sequences on dataset *DNA*. (b) Runtime comparison among the four algorithms for database sizes varying from 1 to 5 K sequences on dataset *Protein*.

BI-Directional Extension and prunes the search space by BackScan pruning strategy. Three typical algorithms, FEAT [17], FSGP [16], and VGEN [15] are developed for mining sequential generators. The first two of them employ a pattern-growth approach by extending the PrefixSpan algorithm. While VGEN performs a depth-first exploration of the search space using a vertical representation of the database, which has a steadily better performance in terms of runtime efficiency. Even with various kinds of enhancements, the above sequential pattern mining algorithms still encounter a challenge that they spawn a rather large pattern set, since the mining process needs to generate an explosive number of smaller subsequences.

From the mining mechanism point of view, the sequential pattern mining algorithms including upper-closed and lower-closed ones can be categorized into A priori-based [15], [32], [34], [35], [40] and pattern-growth [12], [16], [17], [27], [29], [33], [41], [42] ones. The A priori-based algorithms, in order to find all frequent sequential patterns including closed sequential patterns and sequential generators, in each iteration, need to enumerate all possible combinations of frequent subsequences formed in last pass to produce potential longer sequences, which is inherently costly in both runtime and space usage. For example, based on the reports of [43], PrefixSpan takes about 6,000 seconds to enumerate all the 29,711,305 frequent patterns in which only 15,200 patterns are closed when the minimum support is 0.3 percent on the real data set CSLOGS; and the volume of id-lists by SPADE or ISM algorithms is several times larger than that of initial database. Similarly, the classic pattern-growth algorithms begin with a frequent pattern, and grow the pattern when traversing the sequence search space. They often fail to terminate in reasonable time and run out of memory since the daunting number of candidate sub-patterns or frequent sub-trees causes intractable workload. As discussed earlier, both pre-pruning and post-pruning techniques can not be pushed into the closed sequential pattern and sequential generator mining algorithms for performing our mining task. Therefore, previous methods devised for upper-closed or lower-closed sequential pattern mining cannot be applied to contiguous sequential generator mining, which makes such mining a quite challenging task.

Mining sequential generators by imposing the contiguous constraint can achieve a better performance compared to mining upper-closed and lower-closed sequential generator. In addition, the adjacency of patterns has presented convincing arguments that it can minimize the result set and is beneficial for classifying sequences in some particular applications. However, to our best knowledge, no attention has been paid to feeding such feature along with pattern lower-closure property to mining contiguous sequential generators.

Discovering frequent subsequences as sequential patterns in a set of biological sequences is evidence that the patterns occur not by chance but because they share some biological function. For example, the shared biological function which accounts for the similarity of a subset of sequential patterns might be a conserved functional motif. However, a sequential pattern does not necessarily mean a motif. Generally, the set of motifs is only a small subset of the frequent subsequences. If we directly identify the motifs from the big dataset or the growing explosive data scale, the finding process is prohibitively expensive in both runtime and space usage. An nice alternative solution to the problem can be designed: the compact subsequences, such as contiguous sequential generators, are discovered first, and the subsequences are then used to identify the motifs, which significantly reduces computation cost and improves the accuracy of finding motifs.

6 Conclusion

We introduced and studied the problem of mining contiguous sequential generators in biological sequences. We proposed ConSgen to identify contiguous sequential generators, which minimizes the inefficient and redundant patterns and greatly reduces the search space. By using the n-gram model, ConSgen splits the input sequences to generate the pattern candidates, which accurately preserve the items' occurrences in original sequences. And then, it utilizes *redundant snippet pruning*, *max-prefix-suffix pruning* and *support pruning* to prune the unpromising search space. Finally, ConSgen launches a divide-and-conquer technique called equivalence class-based lower-closure checking scenario to efficiently find the sequential generators with contiguous constraint.

We used two real-life data sets with varied densities to study the performance of ConSgen algorithm. Experimental study demonstrated that the set of contiguous sequential generators discovered by ConSgen is much more compact than the set of general sequential patterns, especially when feeding a low support threshold or a pattern-enriched database. Moreover, ConSgen is more efficient in terms of both runtime and memory usage, which is preferable to the state-of-the-art algorithms (VGEN, CloSpan, PrefixSpan, BIDE, FSGP, and FEAT) for big data processing.

Although ConSgen focuses on the discovering of contiguous sequential generators without considering the gapped alignments, it is also feasible to extend this line for gapped sequential generator mining. Thus, ConSgen can be further exploited to discover the binding sites, conserved domains, and functional motifs in biological sequences, which are interesting issues for future research.

Acknowledgments

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 Dr. Liang Tao and Dr. Dingyu Yang for helpful discussions and suggestions during the development of ConSgen and helpful comments on the manuscript. In addition, they would also like to thank the editors and anonymous reviewers. This work was supported by the National Natural Science Foundation of China (NSFC, under the Grant No. 61375053) and the National High Technology Research and Development Program of China (863 Program No. 2009AA04Z106). Y. Wang is the corresponding author.

References

[1] C.-W. Huang, W.-S. Lee, and S.-Y. Hsieh, "An improved heuristic algorithm for finding motif signals in dna sequences," *IEEE/ACM Trans. Comput. Biol. Bioinformat.*, vol. 8, no. 4, pp. 959–975, Jul. 2011.

[2] P. Machanick and T. L. Bailey, "Meme-chip: Motif analysis of large dna datasets," *Bioinformatics*, vol. 27, no. 12, pp. 1696–1697, 2011.

[3] C.-H. Wei, B. R. Harris, H.-Y. Kao, and Z. Lu, "tmvar: A text mining approach for extracting sequence variants in biomedical literature," *Bioinformatics*, p. btt156, 2013.

[4] Z. Yao, K. L. MacQuarrie, A. P. Fong, S. J. Tapscott, W. L. Ruzzo, and R. C. Gentleman, "Discriminative motif analysis of High-throughput dataset," *Bioinformatics*, vol. 30, no. 6, pp. 775–783, 2014.

[5] S. Tanaka, "Improved exact enumerative algorithms for the planted (l, d)-motif search problem," *IEEE/ACM Trans. Comput. Biol. Bioinformat.*, vol. 11, no. 2, pp. 361–374, Mar./Apr. 2014.

[6] K.-S. Leung, K. H. Lee, J.-F. Wang, E. Y. Ng, H. L. Chan, S. K. Tsui, T. S. Mok, P.-H. Tse, and J.-Y. Sung, "Data mining on dna sequences of hepatitis b virus," *IEEE/ACM Trans. Comput. Biol. Bioinformat.*, vol. 8, no. 2, pp. 428–440, Mar./Apr. 2011.

[7] C. Felicioli and R. Marangoni, "Bpmatch: An efficient algorithm for a segmental analysis of genomic sequences," *IEEE/ACM Trans. Comput. Biol. Bioinformat.*, vol. 9, no. 4, pp. 1120–1127, July/Aug. 2012.

[8] T. L. Bailey, "Dreme: Motif discovery in transcription factor Chip-seq data," *Bioinformatics*, vol. 27, no. 12, pp. 1653–1659, 2011.

[9] A. K. Wong and E.-S. A. Lee, "Aligning and clustering patterns to reveal the protein functionality of sequences," *IEEE/ACM Trans. Comput. Biol. Bioinformat.*, vol. 11, no. 3, pp. 548–560, May/Jun. 2014.

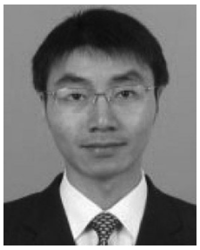
[10] P. Boyen, F. Neven, D. Van Dyck, F. L. Valentim, and A. D. van Dijk, "Mining minimal motif pair sets maximally covering interactions in a protein-protein interaction network," *IEEE/ACM Trans. Comput. Biol. Bioinformat.*, vol. 10, no. 1, pp. 73–86, Jan./Feb. 2013.

[11] J. M. Freire, S. A. Dias, L. Flores, A. S. Veiga, and M. A. Castanho, "Mining viral proteins for antimicrobial and Cell-penetrating drug delivery peptides," *Bioinformatics*, p. btv131, 2015.

[12] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Mining sequential patterns by pattern-growth: The prefixspan approach," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 11, pp. 1424–1440, Nov. 2004.

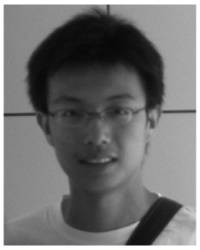
[13] J. Wang, J. Han, and C. Li, "Frequent closed sequence mining without candidate maintenance," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 8, pp. 1042–1056, Aug. 2007.

[14] J. Zhang, Y. Wang, and D. Yang, "Ccspan: Mining closed contiguous sequential patterns," *Knowl.-Based Syst.*, vol. 89, pp. 1–13, 2015.
[15] P. Fournier-Viger, A. Gomariz, M. Šebek, and M. Hlosta, "Vgen: Fast vertical mining of sequential generator patterns," in *Proc. Data Warehousing Knowl. Discovery*, 2014, pp. 476–488.
[16] S. Yi, T. Zhao, Y. Zhang, S. Ma, and Z. Che, "An effective algorithm for mining sequential generators," *Procedia Eng.*, vol. 15, pp. 3653–3657, 2011.
[17] C. Gao, J. Wang, Y. He, and L. Zhou, "Efficient mining of frequent sequence generators," in *Proc. 17th Int. Conf. World Wide Web*, 2008, pp. 1051–1052.
[18] B. Kao, M. Zhang, C.-L. Yip, D. W. Cheung, and U. Fayyad, "Efficient algorithms for mining and incremental update of maximal frequent sequences," *Data Mining Knowl. Discovery*, vol. 10, no. 2, pp. 87–116, 2005.
[19] U. Yun, G. Lee, and K. H. Ryu, "Mining maximal frequent patterns by considering weight conditions over data streams," *Knowl.-Based Syst.*, vol. 55, pp. 49–65, 2014.
[20] J. Zhang, Y. Wang, and H. Wei, "An interaction framework of Service-oriented ontology learning," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage.*, 2012, pp. 2303–2306.
[21] C.-Y. Tsai and B.-H. Lai, "A Location-item-time sequential pattern mining algorithm for route recommendation," *Knowl.-Based Syst.*, vol. 73, pp. 97–110, 2015.
[22] J. Zhang, Y. Wang, and D. Yang, "Automatic learning common definitional patterns from Multi-domain wikipedia pages," in *Proc. IEEE Int. Conf. Data Mining Workshop.*, 2014, pp. 251–258.
[23] D. Lo, S.-C. Khoo, and J. Li, "Mining and ranking generators of sequential patterns," in *Proc. SDM*, 2008, pp. 553–564.
[24] Z. Zeng, J. Wang, J. Zhang, and L. Zhou, "Fogger: An algorithm for graph generator discovery," in *Proc. 12th Int. Conf. Extending Database Technol.: Adv. Database Technol.*, 2009, pp. 517–528.
[25] J. Rissanen, "Minimum description length principle," *Encyclopedia Mach. Learn.*, pp. 666–668, 2010.
[26] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *Proc. 7th Int. Conf. Database Theory*, 1999, pp. 398–416.
[27] X. Yan, J. Han, and R. Afshar, "Clospan: Mining closed sequential patterns in large datasets," in *Proc. SIAM Int. Conf. Data Mining*, 2003, pp. 166–177.
[28] R. Agrawal, R. Srikant, et al., "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases*, 1994, vol. 1215, pp. 487–499.
[29] J. Wang and J. Han, "Bide: Efficient mining of frequent closed sequences," in *Proc. 20th Int. Conf. Data Eng.*, 2004, pp. 79–90.
[30] P. F. Viger, A. Gomariz, T. Gueniche, A. Soltani, C.-W. Wu, and V. S. Tseng, "Spmf: A java open-source pattern mining library," *J. Mach. Learn. Res.*, vol. 15, pp. 3389–3393, 2014.
[31] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proc. 11th Int. Conf. Data Eng.*, 1995, pp. 3–14.
[32] R. Srikant and R. Agrawal, *Mining Sequential Patterns: Generalizations and Performance Improvements*. New York, NY, USA: Springer, 1996.
[33] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu, "Freespan: Frequent pattern-projected sequential pattern mining," in *Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2000, pp. 355–359.
[34] M. J. Zaki, "Spade: An efficient algorithm for mining frequent sequences," *Mach. Learn.*, vol. 42, no. 1-2, pp. 31–60, 2001.
[35] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu, "Sequential pattern mining using a bitmap representation," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2002, pp. 429–435.
[36] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: Current status and future directions," *Data Mining Knowl. Discovery*, vol. 15, no. 1, pp. 55–86, 2007.
[37] J. Pei, J. Liu, H. Wang, K. Wang, S. Y. Philip, and J. Wang, "Efficiently mining frequent closed partial orders," in *Proc. IEEE 13th Int. Conf. Data Mining*, 2005, pp. 753–756.
[38] C. Luo and S. M. Chung, "Efficient mining of maximal sequential patterns using multiple samples," in *Proc. SDM*, 2005, pp. 415–426.
[39] C. Zhang, J. Han, L. Shou, J. Lu, and T. La Porta, "Splitter: Mining Fine-grained sequential patterns in semantic trajectories," *Proc. VLDB Endowment*, vol. 7, no. 9, pp. 769–780, 2014.
[40] M. Zhang, B. Kao, C.-L. Yip, and D. Cheung, "A Gsp-based efficient algorithm for mining frequent sequences," in *Proc. IC-AI*, 2001, pp. 497–503.
[41] F. Guil and R. Marín, "A tree structure for Event-based sequence mining," *Knowl.-Based Syst.*, vol. 35, pp. 186–200, 2012.
[42] G. Pyun, U. Yun, and K. H. Ryu, "Efficient frequent pattern mining based on linear prefix tree," *Knowl.-Based Syst.*, vol. 55, pp. 125–139, 2014.
[43] L. Chang, T. Wang, D. Yang, H. Luan, and S. Tang, "Efficient algorithms for incremental maintenance of closed sequential patterns in large databases," *Data Knowl. Eng.*, vol. 68, no. 1, pp. 68–106, 2009.



Jingsong Zhang received the MSc degree in computer science from the University of Shanghai for Science and Technology. He is currently working toward the PhD degree in the Department of Computer Science and Engineering, Shanghai Jiao Tong University. He will be a postdoctoral fellow in Shanghai Institutes for Biological Sciences, CAS. His research interests include data mining, sequential pattern mining, and bioinformatics. His research is currently supported in part by the National Natural Science Foundation of China (NSFC) and the National High Technology Research and Development Program of China (863 Program). He has been a reviewer for some academic conferences.

Yinglin Wang received the PhD degree in computer science from the Nanjing University of Science and Technology in 1998. His research interests include data mining, knowledge management, and software requirement analysis. He is currently a full professor in the Department of Computer Science and Technology at the Shanghai University of Finance and Economics. Previously, he was a full professor in the Department of Computer Science and Engineering at the Shanghai Jiao Tong University. He has chaired or served in some program committees of international conferences, and has been a reviewer for some academic journals. He is a member of the IEEE Computer Society.



Chao Zhang received both the BS and MS degrees in computer science from Zhejiang University. He is currently working toward the PhD degree in the Department of Computer Science, University of Illinois at Urbana-Champaign. His research interests include spatiotemporal data mining and information network analysis.

Yongyong Shi received the dual bachelor degrees of both biotechnology and international economy and trade, and the PhD degree in biochemistry and molecular biology from Shanghai Jiao Tong University, China, in 2001 and 2006, respectively. He is currently the distinguished professor in the Bio-X Institutes at Shanghai Jiao Tong University. His primary research interest is the genetic studies of complex traits, including mental disorders, tumors, and metabolic diseases. He has published more than 100 papers in leading international journals including corresponding author or first author for seven publications in *Nature Genetics*, one publication in *Nature Nanotechnology*; co-first author or co-author for five publications in *Nature Genetics*, one publication in *Nature*, and one publication in *JAMA*. He has received various awards/honors including the Cheung Kong Scholar of the Ministry of Education of China, the winner of national outstanding youth fund of China, the Author of National Excellent Doctoral Dissertation of China, the Laureates of Tan Jiazhen Life Science Innovation Award, the Leading Scientist of National 973 Project 2015CB559100 of China. He is a co-editor-in-chief of *Hereditas*, member of the editorial board of *Experimental Biology and Medicine*, and on the editorial boards of various other journals.