NOSEP: Nonoverlapping Sequence Pattern Mining With Gap Constraints

Youxi Wu, Yao Tong, Xingquan Zhu, *Senior Member, IEEE*, and Xindong Wu, *Fellow, IEEE*

***Abstract*—Sequence pattern mining aims to discover frequent subsequences as patterns in a single sequence or a sequence database. By combining gap constraints (or flexible wildcards), users can specify special characteristics of the patterns and discover meaningful subsequences suitable for their own application domains, such as finding gene transcription sites from DNA sequences or discovering patterns for time series data classification. Due to the inherent complexity of sequence patterns, including the exponential candidate space with respect to pattern letters and gap constraints, to date, existing sequence pattern mining methods are either incomplete or do not support the *Apriori* property because the support ratio of a pattern may be greater than that of its subpatterns. Most importantly, patterns discovered by these methods are either too restrictive or too general and cannot represent underlying meaningful knowledge in the sequences. In this paper, we focus on a nonoverlapping sequence pattern mining task with gap constraints, where a nonoverlapping sequence pattern allows sequence letters to be flexibly and maximally utilized for pattern discovery. A new *Apriori*-based nonoverlapping sequence pattern mining algorithm, NOSEP, is proposed. NOSEP is a complete pattern mining algorithm, which uses a specially designed data structure, Nettree, to calculate the exact occurrence of a pattern in the sequence. Experimental results and comparisons on biology DNA sequences, time series data, and Gazelle datasets demonstrate the efficiency of the proposed algorithm and the uniqueness of nonoverlapping sequence patterns compared to other methods.**

***Index Terms*—Gap constraint, Nettree, nonoverlapping, pattern matching, sequence pattern mining.**

Manuscript received April 22, 2017; revised July 19, 2017; accepted September 5, 2017. This work was supported in part by the U.S. National Science Foundation under Grant IIS-1613950, in part by the National Natural Science Foundation of China under Grant 61229301, Grant 61370144, and Grant 61673159, and in part by the Graduate Student Innovation Program of Hebei Province under Grant CXZZSS2017037. This paper was recommended by Associate Editor S. Ventura. *(Corresponding author: Youxi Wu.)*

Y. Wu and Y. Tong are with the School of Computer Science and Engineering, Hebei University of Technology, Tianjin 300401, China, also with the School of Economics and Management, Hebei University of Technology, Tianjin 300401, China, and also with the Hebei Province Key Laboratory of Big Data Calculation, Tianjin 300401, China (e-mail: wuc@scse.hebut.edu.cn).

X. Zhu is with the Department of Computer, Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431 USA (e-mail: xzhu3@fau.edu).

X. Wu is with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA 70504 USA (e-mail: xwu@louisiana.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2017.2750691

I. Introduction

FREQUENT pattern mining [1] is one of the popular research areas. Many algorithms have been proposed and kinds of methods have been applied to tackle some issues, such as the *Apriori* strategy, pattern matching strategy, and even evolutionary algorithms [2]. Sequence pattern mining, as an important branch in frequent pattern mining research, aims to discover frequent subsequence patterns in a single sequence or a sequence database [3]. Such patterns are strongly correlated to meaningful events or knowledge within the data and are commonly applied to numerous fields, such as mining customer purchase patterns [4], mining tree-structure information [5], travel-landscape recommendations [6], time-series analysis and prediction [7]–[9], bug repositories [10], sequence classification [11], [12], biological sequence data analysis [13], and temporal uncertainty [14]. Because a sequence pattern consists of multiple pattern letters occurring in a sequential order, it is possible that when pattern occurring in the sequence, two pattern letters may appear in the required sequential order but have different numbers of letters between them.

Formally, a frequent sequence pattern with gap constraints (or flexible wildcards or wildcard gaps) is defined as $P = p_1[\min_1, \max_1]p_2 \cdots [\min_{m-1}, \max_{m-1}]p_m$ [15], [22]. If $\min_1 = \min_2 = \cdots = \min_{m-1} = a$, and $\max_1 = \max_2 = \cdots = \max_{m-1} = b$, it can be called pattern with periodic gap constraints (or periodic wildcard gaps) [16], [17] and P can be written as $p_1p_2 \cdots p_m$ with gap $= [a, b]$. It is worth noting that pattern $P = C[0, 2]G[1, 3]C$ is not considered as a pattern with periodic gap constraints [17], although the size of the second gap $3 - 1 + 1 = 3$ is the same as that of the first $2 - 0 + 1 = 3$. The number of letters between two pattern letters, say p_1 and p_2, forms a gap $[\min_1, \max_1]$ which has strong implications for the actual usage of the sequence patterns. A small gap between pattern letters is too restrictive to find valid patterns, whereas a large gap makes the pattern too general to represent meaningful knowledge within the data.

Because gap constraints allow users to set gap flexibility to meet their special needs, sequence pattern mining with gap constraints has been applied to many fields, including medical emergency identification [23], mining biological characteristics [24], mining customer purchase patterns [25], feature extraction [3], and so on. During the mining process, all existing methods rely on two major steps, generating candidate patterns and counting pattern occurrences, to discover frequent subsequences. However, in a sequence data environment, the occurrence of a pattern in the sequence is inherently complicated because a letter in the sequence may match multiple pattern letters and different matching may result in different frequency-counting results.

To tackle the above challenges, the state-of-the-art methods [3], [26], [28] mainly rely on three different conditions to count the pattern frequency: 1) no-condition [16], [17], [26]; 2) the one-off condition [13], [27]; and 3) the nonoverlapping condition [28]. Compared with the one-off condition and the nonoverlapping condition, some researches do not employ any other condition, to make it easy to understand. In this paper, we call them no-condition (or without condition) which allows a sequence letter to match and rematch any pattern letter. The one-off condition allows a sequence letter being used only once to match a pattern letter and the nonoverlapping condition allows a sequence letter to match and rematch pattern letter as long as the matched patterns letters are different (i.e., nonoverlapping). Therefore, the nonoverlapping condition is less restrictive than the one-off condition and more specific than no-condition for pattern mining. As a result, it is possible to find more meaningful patterns from the sequences.

In addition to the above nonoverlapping condition advantage, compared to the one-off condition, existing sequence pattern mining methods with gaps cannot balance of the *Apriori* property and completeness. On one hand, some methods satisfy the *Apriori* property, but they belong to approximate mining and cannot calculate the exact occurrence of the patterns (by using approximate occurrence counting). As a result, some frequent patterns may be missed, such as sequence pattern mining under the one-off condition [13], [27] or under the nonoverlapping condition [28]. On the other hand, some methods can calculate the exact support of the patterns, but both the support and support ratio of a super-pattern can be greater than its subpattern. In other words, both the support and support ratio do not satisfy anti-monotonicity. Therefore, these issues do not satisfy the *Apriori* property and have to adopt the *Apriori*-like property by expanding the search spaces to achieve completeness mining, such as sequence pattern mining under no-condition [16], [17], [26].

The above observations motivate the proposed research which intends to use the nonoverlapping condition for mining sequence patterns with gap constraints. Compared to the existing sequence pattern mining algorithms, the proposed nonoverlapping sequence pattern (NOSEP) algorithm is an *Apriori*-based mining algorithm that is able to count the exact occurrence of the patterns. Our experiments in Section V demonstrate that NOSEP can discover more meaningful patterns than the state-of-the-art algorithms.

The main contributions of this paper are fourfold.

1) We briefly review the existing sequence pattern mining methods which are either incomplete or do not support the *Apriori* property.
2) We theoretically analyze that NOSEP mining with gap constraints (NOS) could be a complete and *Apriori*-based mining method, therefore it is superior to the state-of-the-art mining methods.
3) We propose NOSEP which employs a pattern matching strategy using Nettree data structure to compute the support of a pattern and adopts a pattern growth strategy to reduce the candidate space for effective sequence pattern mining with gap constraints.
4) Experiments on DNA sequence mining and time series data mining demonstrate that NOSEP can discover more frequent patterns than the state-of-the-art algorithms under the same conditions, and patterns discovered by NOSEP are more effective than those discovered by other competitive algorithms.

The remainder of this paper is organized as follows. Related work is introduced in Section II, followed by the problem definition in Section III. Section IV proposes an effective mining algorithm NOSEP and presents the complexity analysis. Section V reports the algorithm performance and comparisons, and we conclude this paper in Section VI.

II. Related Work

Sequence pattern mining has been widely applied in various fields [23], [29], [32]. But the mining results sometimes cannot fulfill a special request. For example, patterns with gap constraints can be used in many fields, but traditional pattern mining methods fail to solve this challenge. These methods have a variety of forms: 1) no-condition [16], [17], [26]; 2) the one-off condition [13], [27]; and 3) the nonoverlapping condition [28]. In this paper, methods without adding any condition are referred to as no-condition.

In Example 1, we use an example to clarify the relationships of these methods.

Example 1: Given a sequence $S = s_1 s_2 s_3 s_4 s_5 = CGCGC$ and a pattern $P = p_1[\min_1, \max_1]p_2[\min_2, \max_2]p_3 = C[0, 2]G[0, 2]C$, the occurrences of P in S are shown in Fig. 1.

	1	2	3	4	5	
$S =$	C	G	C	G	C	
	C	G	C	.	.	The first occurrence
	C	G	.	.	C	The second occurrence
	C	.	.	G	C	The third occurrence
	.	.	C	G	C	The fourth occurrence

Fig. 1. All occurrences of pattern P in sequence S.

Generally, a group of positions of the pattern in the sequence is used to represent an occurrence. Therefore, the first occurrence can be represented as $\langle 1, 2, 3\rangle$, and others are $\langle 1, 2, 5\rangle$, $\langle 1, 4, 5\rangle$, and $\langle 3, 4, 5\rangle$, and all four occurrences of P will be counted in sequence S under no-condition. When matching pattern letters, some methods employ special conditions, including the one-off condition [13], [27] and the nonoverlapping condition [28]. Both of them consider the relationship between one occurrence and other occurrences.

Under the one-off condition (which is called the stronger version of the nonoverlapping condition as described in [28]), each of the characters in the sequence can be used only once. But under the nonoverlapping condition, s_j cannot be reused by the same p_i and can be reused by other p_i [28]. Therefore, there is only one occurrence, which can be any one selected from the above four occurrences under the one-off condition. For example, if we select $\langle 1, 2, 3\rangle$, $\langle 3, 4, 5\rangle$ is not an occurrence under the one-off condition, since $\langle 3\rangle$ cannot be reused. However, $\langle 1, 2, 3\rangle$ and $\langle 3, 4, 5\rangle$ are two nonoverlapping

TABLE I
SUMMARY AND COMPARISON SOF MAJOR RELATED WORK IN THE FIELD

	Type of research	Type of condition	Periodic gap constraints	Length constraints	Support	Mining type	Pruning strategy
Li et al [3] Zhang et al [17] Zhu and Wu [26] Wu et al [16]	Pattern mining	No-condition	Yes Yes Yes Yes	No No No No	Exact	Exact	Apriori-like
Bille et al [19] Wu et al [33]	Pattern matching	Other/No-condition [1] No-condition	No No	No Yes	Exact Exact	–[2] –	– –
Wu et al [13] Lam et al [27]	Pattern mining	One-off condition	No No	No No	Approximate	Approximate	Apriori
Guo et al [21]	Pattern matching	One-off condition	No	Yes	Approximate	–	–
Ding et al [28]	Pattern mining	Non-overlapping condition	Yes	Yes	Approximate	Approximate	Apriori
Wu et al [22]	Pattern matching	Non-overlapping condition	No	Yes	Exact	–	–
This paper	Pattern mining	Non-overlapping condition	No	Yes	Exact	Exact	Apriori

Note 1: Bille et al [19] proposed two algorithms to calculate the last positions of occurrences and all occurrences, respectively. The former is other kinds of pattern matching while the latter belongs to pattern matching under no-condition.
Note 2: We do not consider mining type and pruning strategy for pattern matching issue.

occurrences under the nonoverlapping condition, since ⟨3⟩ matches p_3 and p_1, respectively.

Because pattern matching strategy plays an important role in the sequence pattern mining task, we summarize and compare the related pattern matching methods in Table I.

To date, researches have primarily focused on sequence pattern mining under no-condition. For example, Zhang *et al.* [17] first addressed sequence pattern mining with periodic gap constraints and proposed an effective algorithm to tackle the problem. Zhu and Wu [26] proposed a more effective algorithm which can be used to deal with a set of sequences. Our previous research [16] employed Nettree data structure and proposed mining sequential pattern using incomplete Nettree with depth first search (MAPD) algorithm which is more effective than the previous the state-of-the-art algorithms. Li *et al.* [3] showed that this kind of mining method can be employed for feature selection for the purpose of classification. All the above mining methods belong to no-condition. The advantage of sequence pattern mining under no-condition lies that under no-condition the support of P in S can be calculated exactly according to many previous researches [16], [17], [26]. Many researches also focused on pattern matching under no-condition. These researches are beyond the limitations of periodic gap constraints [19], [33], exact pattern matching [34], and positive gap constraints [35]. But the disadvantage of sequence pattern mining under no-condition lies that the support (the number of occurrences) of a pattern is no less than that of its subpattern under no-condition. For example, the supports of $T[0, 2]C$ and $T[0, 2]C[0, 2]A$ in TTCCAA are 4 and 8, respectively. Therefore, this mining method does not meet the *Apriori* property. Although some other researches under no-condition, such as [7] and [8], also meet the *Apriori* property but they change the definitions and the mining results are a little bit different, therefore, these researches can be seen as approximate mining. In general, sequence pattern mining with no-condition can exactly find the patterns but fails to meet the *Apriori* property.

The advantage of sequence pattern mining under the one-off condition and the nonoverlapping condition is that these two methods meet the *Apriori* property, since support of a pattern is no greater than that of its subpattern. Many researchers focused on pattern matching under the one-off condition. For example, without user-specified gap constraints, Wu *et al.* [13] employed the one-off condition and proposed an effective algorithm to discover pattern from biological sequences. Lam *et al.* [27] proposed two minimum description length-based algorithms for mining nonredundant sets of sequential patterns. Although Lam *et al.* did not use the one-off condition term, actually, the study is a pattern mining under the one-off condition. Unfortunately, calculating the support under the one-off condition is an NP-hard problem [36]. Although many heuristic algorithms [20], [21], [37]–[39] were proposed to calculate the support, all these algorithms cannot calculate the support exactly. Therefore, the disadvantage of sequence pattern mining under the one-off condition lies that some of the frequent patterns may be lost.

Ding *et al.* [28] first addressed the sequence pattern mining under the nonoverlapping condition and proposed GSgrow and CloGSgrow to mine the frequent patterns and the closed frequent patterns, respectively. To the best of our knowledge, they are the only two algorithms considering the nonoverlapping condition for pattern mining. However, some frequent patterns may not be found because the two algorithms employ the INSgrow procedure to calculate the support, which may result in the loss of some feasible occurrences. The reason is shown in challenge 1 in subsection IV-A. Therefore, GSgrow and CloGSgrow are approximate mining algorithms. Another most relevant research is one of our previous works [22] which dealt with a pattern matching issue. In that research, we mainly focused on proving the issue is in P and provided an effective calculating method. This paper however, handles a sequence pattern mining issue. In this paper we show that NOS is superior to other mining methods and propose an effective completeness and *Apriori*-based mining algorithm. Besides [28] and our previous work [22], another most relevant research is our previous work [16] which also mined the frequent sequence patterns under no-condition while this paper deals with the patterns under the nonoverlapping condition. Moreover, in [16], we employed Incomplete Nettree structure, the last level of a Nettree, to calculate the support of a pattern while in this paper we adopt Nettree structure. Another difference lies that we will show that pattern growth approach is a more effective pruning strategy to tackle the issue than that in [16]. Therefore, this paper differs from the previous studies.

In summary, existing sequence pattern mining with gap constraints cannot achieve a balance between the *Apriori* property and completeness. In this paper, we investigate NOS, with the aim of discovering the complete set of frequent sequence patterns based on the *Apriori* property.

III. Problem Definition

Definition 1 (Pattern With Gap Constraints and Sequence): Pattern P with gap constraints can be described as $p_1[\min_1, \max_1]p_2 \cdots [\min_{m-1}, \max_{m-1}]p_m$, where $p_j \in \Sigma$, $\min_j$ and $\max_j$ are two nonnegative integers and represent the minimum gap constraint and maximum gap constraint, respectively, $1 \le j \le m$, and Σ is a set of all event items. Given the gap constraints gap $= [a, b]$, pattern P with periodic gap constraints can be written as $p_1[a, b]p_2 \cdots [a, b]p_m$ or $P = p_1p_2 \cdots p_m$ with gap $= [a, b]$, where $0 \le a \le b$. Sequence S with length n can be written as $s_1s_2 \cdots s_i \cdots s_n$, where $1 \le i \le n$ and $s_i \in \Sigma$. The number of elements in the set is denoted by $|\Sigma|$.

For example, in a DNA sequence, Σ is {A, T, C, G} and $|\Sigma|$ is 4.

Definition 2 (Occurrence): A group of m integers $L = \langle l_1, l_2, \ldots, l_m\rangle$ is called an occurrence of P in S, if and only if $1 \le l_1 < l_2 < \cdots < l_m \le n$, $\min_j \le l_{j+1} - l_j - 1 \le \max_j$, $p_1 = s_{l_1}$,$p_2 = s_{l_2}$,..., and $p_m = s_{l_m}$.

Example 2: Suppose sequence $S = s_1s_2s_3s_4s_5 = CGCGC$ and pattern $P = C[0, 2]G[0, 2]C$ are given, all occurrences of P in S are: $\langle 1, 2, 3\rangle$, $\langle 1, 2, 5\rangle$, $\langle 1, 4, 5\rangle$, and $\langle 3, 4, 5\rangle$, while all occurrences of CGC with gap $= [0, 1]$ in S are $\langle 1, 2, 3\rangle$ and $\langle 3, 4, 5\rangle$.

Definition 3 (Length Constraints): The length constraints can be written as len = [minlen, maxlen], where minlen and maxlen are the minimum length constraint and the maximum length constraint, respectively. If $L = \langle l_1, l_2, \ldots l_m\rangle$ satisfies minlen $\le l_m - l_1 + 1 \le$ maxlen, then L is an occurrence with length constraints.

We can see that all occurrences with length constraints len $= [1, 4]$ of $P = C[0, 2]G[0, 2]C$ in S are $\langle 1, 2, 3\rangle$, $\langle 3, 4, 5\rangle$ in Example 2. For example, for occurrence $\langle 3, 4, 5\rangle$ $5 - 3 + 1 = 3$ satisfies len $= [1, 4]$.

Definition 4 (Non-Overlapping Occurrence Set and Support): Let $L = \langle l_1, l_2, \ldots, l_m\rangle$ and $L' = \langle l'_1, l'_2, \ldots, l'_m\rangle$ be two occurrences. If and only if $\forall 1 \le j \le m : l_j \ne l'_j$, L and L' are two nonoverlapping occurrences. If any two occurrences in a set are nonoverlapping, then the set is called nonoverlapping occurrence set. The support of P in S under the nonoverlapping condition, which is denoted by sup(P, S), is the size of the maximum nonoverlapping occurrence set.

In Example 2, all occurrences of P with gap constraints gap $= [0, 2]$ and length constraint len $= [1, 5]$ in S are $\langle 1, 2, 3\rangle$, $\langle 1, 2, 5\rangle$, $\langle 1, 4, 5\rangle$, and $\langle 3, 4, 5\rangle$. The maximum nonoverlapping occurrence set is $\{\langle 1, 2, 3\rangle, \langle 3, 4, 5\rangle\}$. Therefore, under the nonoverlapping condition, sup(P, S) is 2. In particular, we do not consider the relationship between an occurrence and its suboccurrence. For example, we do not consider the relationship between $\langle 1, 2, 3\rangle$ and $\langle 1, 2\rangle$.

Definition 5 (Support of Sequence Database): A set of sequences is called a sequence database, denoted by SDB, $SDB = \{S_1, S_2, \ldots, S_N\}$, where N is the size of sequence database. The support of pattern P in SDB is the sum of supports of P in $S_1, S_2, \ldots, S_N$, denoted by $sup(P, SDB) = \sum_{k=1}^{N} sup(P, S_k)$.

Example 3: Suppose $SDB = \{S_1 = s_1s_2s_3s_4s_5 = CGCGC, S_2 = s_1s_2s_3s_4s_5 = CGTCA\}$. Then the support of pattern P=CGC with gap constraints gap $= [0, 2]$ and length constraint len $= [1, 4]$ in SDB is 3, since its supports in S_1 and S_2 are 2 and 1, respectively.

Definition 6 (Frequent Pattern and NOS): If the support of pattern P in sequence S or in sequence database SDB is no less than the given minimum support threshold minsup, then pattern P is called the frequent pattern. The goal of NOS is to mine all the frequent patterns with the gap and length constraints in sequence S or in sequence database SDB.

Example 4: If the minimum support threshold minsup $= 3$, then pattern $P =$ CGC with gap $= [0, 2]$ and len $= [1, 4]$ is a frequent pattern in Example 4, since its support in SDB is 3. The supports of pattern CGCG in S_1 and S_2 are 1 and 0, respectively. Therefore, the support of pattern CGCG in the SDB is 1. Hence, pattern CGCG is not a frequent pattern.

IV. NOSEP Algorithm

For sequence pattern mining, the two major factors impacting on the mining performance include: calculation of the support and reduction of the candidate pattern space. Accordingly, in subsection IV-A we propose an effective algorithm, named NETGAP, to calculate the support and we prove the completeness of the algorithm. Subsection IV-B proposes the mining algorithm to reduce the candidate pattern space. Moreover, we show the space and time complexities in subsection IV-C and briefly introduce the principles of three competitive algorithms in subsection IV-D.

A. NETGAP

When calculating the pattern support, there are two major challenges: 1) checking pattern occurrence without using a backtracking strategy and 2) distinguishing characters in the sequence that may be reused for pattern matching.

Challenge 1: The information of the occurrence of a subpattern cannot be employed directly to calculate the occurrence for the super-pattern, because it may cause some feasible occurrences to be lost. For instance, given $S = s_1s_2s_3s_4s_5 =$ ATTGC, the first nonoverlapping occurrence of subpattern $A[0, 1]T$ is $\langle 1, 2\rangle$. But there is no nonoverlapping occurrence of super-pattern $P = A[0, 1]T[0, 1]C$ based on $\langle 1, 2\rangle$, since $\langle 1, 2, 5\rangle$ does not satisfy the gap constraints. We know that $\langle 1, 3, 5\rangle$ is a nonoverlapping occurrence of super-pattern $P = A[0, 1]T[0, 1]C$. INSgrow [28] calculates the occurrences for a pattern based on the suboccurrences for its subpattern. INSgrow, without using backtracking strategy, cannot find the occurrence $\langle 1, 3, 5\rangle$, therefore, may lose some feasible occurrences. However, if a backtracking strategy is employed, then $\langle 1, 3, 5\rangle$ can be found based on $\langle 1, 2\rangle$. Apparently, this method is less effective. Therefore, an effective algorithm without using a backtracking strategy should be presented.

Challenge 2: When dealing with the nonoverlapping condition, after finding an occurrence, we cannot use an unmatchable character "*X*" to replace the corresponding character in the sequence. For instance, we know that the first occurrence in Example 1 is $\langle 1, 2, 3\rangle$. If X is used to replace the corresponding character in the sequence, the new sequence is "*XXXGC*", and then nonoverlapping occurrence $\langle 3, 4, 5\rangle$ cannot be obtained. Hence, the algorithm should effectively distinguish which characters in the sequence can be reused.

To tackle the above challenges, we propose to use the Nettree data structure to solve the problem.

Definition 7 (Nettree): Nettree [33] is similar to a tree data structure, consisting of root, leaf, level, parent, child, and so on. Nevertheless, Nettree has three characteristics that are evidently different from the tree structure.

1) A Nettree may have n roots, where $n > 1$.
2) To describe a node effectively, node i in the jth level is denoted by n_j^i since the same node label can occur on different levels.
3) Any node except the root may have more than one parent and all its parents must be at the same level; that is the nonroot node n_j^i $(j > 1)$ may have multiple parents $\{n_{j-1}^{i_1}, n_{j-1}^{i_2}, \ldots, n_{j-1}^{i_m}\}$ $(m \geq 1)$, and thus there may be multiple paths from a node to a root node.

Definition 8 (Absolute Leaf): A leaf in the mth level Nettree is called an absolute leaf.

Definition 9 (Full Path): A path from a root to an absolute leaf in a Nettree is called a full path.

Lemma 1: Each occurrence of pattern P in sequence S can be represented as a full path in the Nettree and a full path corresponds to an occurrence.

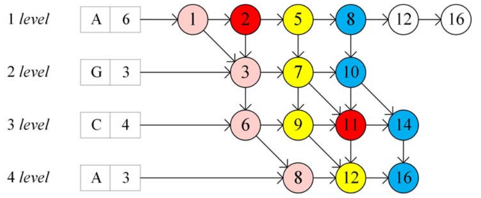
Proof: Our previous work [33] has shown that all occurrences of pattern P in sequence S can be transformed into a Nettree and each full path corresponds to an occurrence; that is, each occurrence can be represented as a full path in the Nettree. ■

Definition 10 (Minimal Full Path, the Minimal Occurrence, the Maximal Full Path, and the Maximal Occurrence): A full path that iterates the leftmost child from the min-root to its leaf is called the minimal full path. The corresponding occurrence of the minimal full path is called the minimum occurrence. Similarly, a full path that iterates the rightmost parent from the max-leaf to its root is called the maximal full path and its corresponding occurrence is called the maximal occurrence.

Lemma 2: Let A and B be two full paths that do not contain the same Nettree node. The corresponding occurrences of A and B are the nonoverlapping occurrences.

Proof: A and B are $\langle n_1^{a_1}, n_2^{a_2}, \ldots, n_m^{a_m}\rangle$ and $\langle n_1^{b_1}, n_2^{b_2}, \ldots, n_m^{b_m}\rangle$, respectively. For all $i (1 \leq i \leq m)$, a_i is not equal to b_i, since A and B do not contain the same node. Therefore, $\langle a_1, a_2, \ldots, a_m\rangle$ and $\langle b_1, b_2, \ldots, b_m\rangle$ are two nonoverlapping occurrences. ■

Nettree data structure is suitable to solve this problem effectively. First, a Nettree can be created according to the pattern and the sequence. Then we obtain a minimal full path on the Nettree and prune the minimal full path and other invalid nodes from the Nettree. We iterate the above process until the Nettree is NULL. Example 5 illustrates the processes and concepts of Definitions 8–10.



Note: pink nodes, yellow nodes, and blue nodes in the figure represent three full paths of nodes, respectively; red nodes represent invalid nodes after pruning a minimal full path.

Fig. 2. Corresponding Nettree of Example 5.

Example 5: Given a sequence $S = s_1s_2s_3s_4s_5s_6s_7s_8s_9 s_{10}s_{11}s_{12}s_{13}s_{14}s_{15}s_{16} = AAGTACGACGCATCTA$ and a pattern $P = A[0, 3]G[0, 3]C[0, 3]A$, a Nettree shown in Fig. 2 can be created using our previous CreateNettree algorithm [22]. n_4^8, n_4^{12}, and n_4^{16} are three absolute leaves in the Nettree. Path $\langle n_1^5, n_2^7, n_3^9, n_4^{12}\rangle$ is a full path. Root n_1^1 is the min-root at the beginning. Path $\langle n_1^1, n_2^3, n_3^6, n_4^8\rangle$ is the minimal full path and its corresponding occurrence $\langle 1, 3, 6, 8\rangle$ is the minimal occurrence. Similarly, n_4^{16} is the rightmost leaf. Path $\langle n_1^8, n_2^{10}, n_3^{14}, n_4^{16}\rangle$ is the maximal full path and its corresponding occurrence $\langle 8, 10, 14, 16\rangle$ is the maximal occurrence.

We can obtain the first minimal occurrence $\langle 1, 3, 6, 8\rangle$, marked in pink in Fig. 2. When $\langle 1, 3, 6, 8\rangle$ is pruned from the Nettree, we know that node n_1^2 has no child and is an invalid node. Hence, n_1^2 can also be pruned and marked in red. Then the second minimal occurrence $\langle 5, 7, 9, 12\rangle$, marked in yellow, can be found. After pruning $\langle 5, 7, 9, 12\rangle$, node n_3^{11} has no child and can also be pruned. Finally, the third minimal occurrence $\langle 8, 10, 14, 16\rangle$, marked in blue, is obtained. Therefore, we get three nonoverlapping occurrences of P in S; that is, $sup(P, S) = 3$.

Lemma 3: A node without child nodes can be safely pruned.

Proof: If n_j^i is a nonabsolute leaf node in a Nettree, there is no path from it to an absolute leaf. Therefore, n_j^i is an invalid node and should be pruned. After pruning n_j^i, we should also check its parents to find out whether or not those nodes have a child. A node that has no child should also be pruned. ■

An algorithm, named NETGAP, which computes the support of P in S, $sup(P, S)$, is shown in Algorithm 1. Since NETGAP will be employed by sequence pattern mining algorithms, when $sup(P, S)$ is greater than the minsup, it is not necessary to continue to calculate the support.

Lemma 4: Let $\langle x_j, x_{j+1}\rangle$ and $\langle y_j, y_{j+1}\rangle$ be two suboccurrences of subpattern $p_j[a_j, b_j]p_{j+1}$. Supposing that $x_j < y_j$ and $x_{j+1} > y_{j+1}$, we can safely say that $\langle x_j, y_{j+1}\rangle$ and $\langle y_j, x_{j+1}\rangle$ are also two suboccurrences.

Proof: Our previous work [22] has shown that if $\langle x_j, x_{j+1}\rangle$ and $\langle y_j, y_{j+1}\rangle$ are two suboccurrences, then $\langle x_j, y_{j+1}\rangle$ and $\langle y_j, x_{j+1}\rangle$ are also two suboccurrences, where $x_j < y_j$ and $x_{j+1} > y_{j+1}$. ■

Algorithm 1 NETGAP

```
Input: Sequence $S$, Pattern $P$, $gap = [a, b]$, $len = [minlen, maxlen]$, and $minsup$
Output: $sup(P, S)$
Create a nettree of $P$ in $S$;
Prune nodes without child nodes (per Lemma 3);
for each $n_1^i$ in nettree do
    $node[1] \leftarrow n_1^i$;        //node used to store an occurrence;
    for $j$=1 to $nettree.level - 1$ step 1 do
        $node[j+1] \leftarrow$ the leftmost child meeting the length constraints of $node[j]$;
    end for
    $sup(P, S) \leftarrow sup(P, S) + 1$ ;
    if $sup(P, S) > minsup$ return $sup(P, S)$;
    Prune nodes without child nodes (per Lemma 3);
end for
return $sup(P, S)$;
```

Example 6: Suppose sequence $S = s_1s_2s_3s_4s_5s_6 = AATTCC$ and pattern $P = A[0, 2]T[0, 2]C$ are given. We can see that $\langle 3, 6\rangle$ and $\langle 4, 5\rangle$ are two suboccurrences of subpattern T[0,2]C. According to Lemma 4, we can safely say $\langle 3, 5\rangle$ and $\langle 4, 6\rangle$ are two suboccurrences.

Theorem 1: The algorithm NETGAP is complete.

Proof: Our previous work [22] has shown that we can obtain a maximal nonoverlapping set D that has k nonoverlapping occurrences; that is, $\langle d_{1,1},\ d_{1,2}, \ldots d_{1,m}\rangle$, $\langle d_{2,1},\ d_{2,2}, \ldots d_{2,m}\rangle, \ldots \langle d_{k,1},\ d_{k,2}, \ldots d_{k,m}\rangle$, where $d_{h,j} < d_{h+1,j}$ and $1 \leq h < k$. That work has also shown that $\langle d_{k,1}, d_{k,2}, \ldots d_{k,m}\rangle$ can be replaced by the maximal occurrence $\langle f_{k,1}, f_{k,2}, \ldots f_{k,m}\rangle$. Now we will show that $\langle d_{1,1}, d_{1,2}, \ldots d_{1,m}\rangle$ can be replaced by the minimal occurrence $\langle g_{1,1}, g_{1,2}, \ldots g_{1,m}\rangle$.

Suppose $\langle d_{1,1}, d_{1,2}, \ldots d_{1,m}\rangle$ is the minimal occurrence, which means that $\langle d_{1,1}, d_{1,2}, \ldots d_{1,m}\rangle$ is the same as $\langle g_{1,1}, g_{1,2}, \ldots g_{1,m}\rangle$. Now $\langle d_{1,1}, d_{1,2}, ...d_{1,m}\rangle$ is different from the minimal occurrence $\langle g_{1,1}, g_{1,2}, \ldots g_{1,m}\rangle$. There are only three cases.

Case 1: There exist j $(1 \leq j \leq m)$ that satisfy $d_{1,j} < g_{1,j}$. $\langle g_{1,1}, g_{1,2}, \ldots g_{1,m}\rangle$ is not a minimal occurrence. This contradicts the assumption that $\langle g_{1,1}, g_{1,2}, \ldots g_{1,m}\rangle$ is a minimal occurrence.

Case 2: For all j $(1 \leq j \leq m)$ $d_{1,j}$ is greater than $g_{1,j}$; that is, $d_{1,j} > g_{1,j}$. $\langle d_{1,1}, d_{1,2}, \ldots d_{1,m}\rangle$ and $\langle g_{1,1}, g_{1,2}, \ldots g_{1,m}\rangle$ are two nonoverlapping occurrences. So there should be $k + 1$ nonoverlapping occurrences. This contradicts the assumption that there are k nonoverlapping occurrences.

Case 3: For all j $(1 \leq j \leq m)$ $d_{1,j}$ is no less than $g_{1,j}$; that is, $d_{1,j} \geq g_{1,j}$. Since $\langle d_{1,1}, d_{1,2}, \ldots d_{1,m}\rangle$ and $\langle d_{i,1}, d_{i,2}, \ldots d_{i,m}\rangle$ $(1 < i \leq k)$ are two nonoverlapping occurrences and $d_{i,j} > d_{1,j}$, $d_{i,j}$ is greater than $g_{i,j}$; that is, $d_{i,j} > g_{1,j}$. Therefore, $\langle g_{1,1}, g_{1,2}, \ldots g_{1,m}\rangle$ and $\langle d_{i,1}, d_{i,2}, \ldots d_{i,m}\rangle$ are two nonoverlapping occurrences. Hence, $\langle g_{1,1}, g_{1,2}, \ldots g_{1,m}\rangle$ can be used to replace $\langle d_{1,1}, d_{1,2}, \ldots d_{1,m}\rangle$.

To sum up, no matter whether $\langle d_{1,1}, d_{1,2}, \ldots, d_{1,m}\rangle$ is the minimal occurrence or not, $\langle d_{1,1}, d_{1,2}, \ldots d_{1,m}\rangle$ can be safely replaced by $\langle g_{1,1}, g_{1,2}, \ldots g_{1,m}\rangle$. As we know NETGAP iterates to find the minimal occurrences. Therefore, the algorithm NETGAP is complete. ■

B. NOSEP Algorithm

Before NOSEP is presented, some related concepts are given at first.

Definition 11 (Prefix and Suffix Subpattern and Super-Pattern): Given a sequence $P = p_1p_2 \cdots p_m$ and event items a and b, if $Q = Pa$, P is called the prefix subpattern of Q and is denoted by $\text{prefix}(Q) = P$. Q is a super-pattern of P. Similarly, if $R = bP$, P is called the suffix subpattern of R and denoted by $\text{suffix}(R) = P$ and $R = bP$. Since $\text{prefix}(Q) = \text{suffix}(R) = P$, R, and Q can be connected to a super-pattern T whose length is $m + 2$ using the operator $\oplus$; that is, $T = Q \oplus R = bPa$.

Example 7: Let pattern P be $ACCT$. The prefix subpattern and the suffix subpattern of P are ACC and CCT, respectively. If $Q = CCTG$, then $T = P \oplus Q = ACCTG$.

Theorem 2: NOS satisfies the *Apriori* property.

Proof: The prefix subpattern and the suffix subpattern of pattern P are Q and R, respectively. S is a given sequence. It can be easily seen that $sup(Q, S) \geq sup(P, S)$ and $sup(R, S) \geq sup(P, S)$ according to Definition 4. So if Q is not a frequent pattern; that is, $sup(Q, S) <$ minsup, then $sup(P, S)$ is less than minsup. Hence, P is also not a frequent pattern as a result of $\text{sup}(P, S) <$ minsup. Similarly, if R is not a frequent pattern, then P is also not a frequent pattern. Obviously, the above cases are still valid in a sequence database. Therefore, NOS satisfies the *Apriori* property. ■

Our previous work [16] employed both breadth-first search and depth-first search to mine the frequent patterns with gap constraints. Since there is no condition in that research, mining sequential pattern using incomplete Nettree with breadth first search (MAPB) and MAPD [16] can calculate all the candidate patterns with the same prefix pattern in one-way scanning the sequence or *SDB*. Therefore, MAPB and MAPD are two effective mining algorithms. However, in this paper, we calculate the support under the nonoverlapping condition. NETGAP cannot calculate the supports of the candidate patterns with the same prefix pattern in one-way scanning the sequence, can calculate the support of a candidate pattern in once scanning. Neither breadth-first search nor depth-first search is a very effective strategy. The following example illustrates this point of view.

Example 8: We find all frequent patterns from sequence $S = s_1s_2s_3s_4s_5s_6s_7s_8s_9s_{10}s_{11}s_{12}s_{13}s_{14}s_{15}s_{16} =$ AAGTACGACGCATCTA, minsup $= 3$, gap constraints gap $= [0, 3]$, and length constraints len $= [1, 15]$.

We know that all the frequent patterns with length 1 are {“A,” “C,” “G,” “T”}, and seven kinds of frequent patterns with length 2 {“AA,” “AC,” “AG,” “CA,” “CC,” “GA,” “GC”}. Hence, the number of candidate patterns with length 3 is $7 \times 4 = 28$, since each frequent pattern with length 2 will generate four kinds of candidate patterns. We can safely know that pattern “AAT” is not a frequent pattern, since pattern “AT” is not a frequent sequence. Therefore, there are seventeen kinds of candidate patterns with length 3 using pattern growth approach; that is, {“AAA,” “AAC,” “AAG,” “ACA,” “ACC,” “AGA,” “AGC,” “CAA,” “CAC,” “CAG,” “CCA,” “CCC,”

Algorithm 2 NOSEP: Mining All the Frequent Patterns Based on Pattern Growth Approach

```
Input: Sequence database $SDB$, $minsup$, $gap = [a, b]$, $len = [minlen, maxlen]$
Output: The frequent patterns in $meta$
 1: Scan sequence database $SDB$ once, calculate the support of each event item, and store the frequent patterns with length 1 into a queue $meta[1]$;
 2: $len \leftarrow 1$;
 3: $C \leftarrow$ gen_candidate($meta[len]$);      // generate candidate set $C$
 4: while $C <> null$ do
 5:     for each $cand$ in $C$ do
 6:         $cand_{sup} \leftarrow 0$;
 7:         $sup_{needed} \leftarrow minsup$;
 8:         for each sequence $s_k$ in $SDB$ do
 9:             $sup \leftarrow$ NETGAP($cand$, $sup_{needed}$);
10:             $sup_{needed} \leftarrow sup_{needed} - sup$;
11:             $cand_{sup} \leftarrow cand_{sup} + sup$;
12:             if $cand_{sup} \geq minsup$ then
13:                 $meta[len + 1]$.enqueue($cand$);
14:                 break;
15:             end if
16:         end for
17:     end for
18:     $len \leftarrow len + 1$;
19:     $C \leftarrow$gen_candidate($meta[len]$);
20: end while
21: return $meta[1] \cup meta[2] ... \cup meta[len]$;
```

“GAA,” “GAC,” “GAG,” “GCA,” “GCC”}. Hence, this example shows that the pattern growth approach is more effective than breadth-first search and depth-first search approaches.

Since the frequent pattern set with a length of $n - 1$ is an ordered set, the candidate pattern set with a length of n is also an ordered set. The following example is used to illustrate how to generate the candidate pattern set with length n based on the pattern growth approach.

Example 9: Let the frequent patterns with length 2, C_2 be {AA, AC, AG, CA, CC, GA, GC}. We will generate the candidate patterns with length 3 based on the pattern growth approach.

First, we get the first pattern in C_2 which is AA. We know that the suffix pattern of AA is “A.” Now, the binary search strategy is employed to find the first position in C_2 whose prefix pattern is also A. We know that the first pattern in C_2 is AA and its prefix pattern is A. So the first candidate pattern AAA is generated. Since AA, AC, and AG have the same parent in the set enumeration tree, A, AAC, and AAG can be obtained. Now, the next pattern of AG is CA and its prefix pattern is “C” which is different from A. The suffix pattern A should be changed. The above steps are iterated until pattern C as the suffix pattern of GC has been processed and all the candidate patterns can be found.

In Algorithm 2, we report the detailed procedures of the NOSEP algorithm. It is worth noting that NOSEP employs an early termination, on steps 11 and 12, to quickly determine frequent patterns. When the support of a pattern is greater than minsup, NOSEP stops calculating the support to speed up the process.

Algorithm 3 gen_candidate($meta[len]$)

```
Input: $meta[len]$
Output: Candidate set $C$
start ← 1;
for $i = 1$ to $|meta[len]|$ do
    $R \leftarrow suffix(meta[len][i])$;
    $Q \leftarrow prefix(meta[len][start])$;
    if $R <> Q$ then
        $start \leftarrow$ binarysearch($meta[len]$, $R$, 1, $|meta[len]|$);
    end if
    if $start >= 1$ && $start <= |meta[len]|$ then
        while $Q == R$ do
            $C$.enqueue($R \oplus Q$);
            $start \leftarrow start + 1$;
            if $start > |meta[len]|$ then
                $start \leftarrow 1$;
                $Q \leftarrow$ prefix($meta[len][start]$);
            end if
        end while
    end if
end for
return $C$;
```

Algorithm gen_candidate, shown in Algorithm 3, is used to generate a candidate set with length len + 1.

C. Complexity Analysis

Theorem 3: The space complexity of NOSEP is $O(M \times (n \times w + L))$ in the worst case and $O(M \times (n \times w/r/r + L))$ in the average case, where M, n, w, L, and r are the maximal length of mined pattern, the maximal length of sequence in the sequence database SDB, $b-a+1$, the number of the candidate patterns, and the size of Σ, respectively.

Proof: Suppose there are L candidate patterns. Apparently, the number of the frequent patterns should be less than L. The maximal lengths of candidate pattern and mined pattern are $O(M)$. So the space complexity of frequent patterns and candidate patterns is $O(M \times L)$. Now, we consider the space complexity of Algorithm 1. It calculates the support of a pattern in a sequence using a Nettree. In the worst case, a Nettree has no more than M levels, each level has no more than n nodes, and each node has no more than w children. Therefore, the space and time complexities of creating a Nettree are both $O(M \times n \times w)$. Hence, the space complexity of NOSEP is $O(M \times (n \times w + L))$ in the worst case. Moreover, in the average case, each level has no more than n/r nodes and each node has no more than w/r children. Hence, the space complexities of Algorithm 1 and NOSEP are $O(M \times n \times w/r/r)$ and $O(M \times (n \times w/r/r + L))$ in the average case, respectively. ■

Theorem 4: The time complexity of NOSEP is $O(M \times M \times N \times w \times L)$ in the worst case and $O(M \times M \times N \times w \times L/r/r/r)$ in

TABLE II
SUMMARY OF THE BENCHMARK DATASETS

Dataset	Type	From	$\vert\Sigma\vert$	Number of sequences	Same size for each sequence	Length
TSS	Human genes[1]	Transcriptional Start Sites	4	100	Yes, 100/sequence	10,000
WTC	Convert sequence [2]	WormsTwoClass in UCR time series data	20	77	Yes, 150/sequence	11,550
DNA1	DNA [3]	Homo Sapiens AL158070	4	1	Single	6,000
DNA2	DNA	Homo Sapiens AL158070	4	1	Single	8,000
DNA3	DNA	Homo Sapiens AL158070	4	1	Single	10,000
DNA4	DNA	Homo Sapiens AL158070	4	1	Single	12,000
DNA5	DNA	Homo Sapiens AL158070	4	1	Single	14,000
DNA6	DNA	Homo Sapiens AL158070	4	1	Single	16,000
SDB1	Protein [4]	ASTRAL95_1_161	20	507	No	91,875
SDB2	Protein	ASTRAL95_1_161	20	338	No	62,985
SDB3	Protein	ASTRAL95_1_161	20	169	No	32,503
SDB4	Protein	ASTRAL95_1_171	20	590	No	109,424
SDB5	Protein	ASTRAL95_1_171	20	400	No	73,425
SDB6	Protein	ASTRAL95_1_171	20	200	No	37,327
BMS1	Gazelle [5]	Clickstream data from an e-commerce	497	59,601	No	149,638
BMS2	Gazelle	Clickstream data from an e-commerce	3,340	77,512	No	358,278

Note 1: TSS, from http://dbtss.hgc.jp/, which was studied in [24], contains 200 human genes with the positive class and the negative class. In this paper, we select the first 100 sequences (positive class).
Note 2: WormsTwoClass in UCR time series data [40] (http://www.cs.ucr.edu/~eamonn/time_series_data/) is converted to demonstrate the capability of the discovered patterns for classification using SAX [41] (http://cs.gmu.edu/~jessica/sax.htm) with parameters $data\_len = 900$, $nseg = 150$, and $alphabet\_size = 20$.
Note 3: Homo Sapiens AL158070 can be downloaded from http://www.ncbi.nlm.nih.gov/nuccore/AL158070.11.
Note 4: ASTRAL95_1_161 and ASTRAL95_1_171 were studied in [42].
Note 5: Gazelle datasets, provided by Fournier-Viger P et al [43], had been used by [28].

the average case, where M, w, L, and r are given in Theorem 3, N is the length of SDB.

Proof: Suppose there are L candidate patterns. Apparently, $|meta[len]|$ is not greater than L. Therefore, the complexity of line 6 in Algorithm 3 is less than $O(\log L)$ since it is a binary search. The time complexity of generating all frequent patterns is $O(L \times \log L)$. We consider the time complexity of NOSEP in the worst case at first. As mentioned in Theorem 3, we know that the time complexity of creating Nettrees is $O(M \times N \times w)$ for an N-length sequence database. Suppose the depth of the Nettree is M. No more than w nodes in the $M-1$th level could be pruned using Lemma 3 since each node has no more than w parents. Similarly, no more than $2 \times w$ nodes in the $M-2$th level could be pruned. Hence, there are no more than $O(M \times M \times w)$ nodes could be deleted. There are no more than N nonoverlapping occurrences. Therefore, the time complexity of lines 8 to 17 in NOSEP is $O(M \times N \times w + M \times M \times N \times w) = O(M \times M \times N \times w)$. Since there are L candidate patterns, in the worst case, the time complexity of NOSEP is $O((M \times M \times N \times w + \log L) \times L) = O(M \times M \times N \times w \times L)$. Moreover, as mentioned in the space complexity, the time complexity of NOSEP is $O(M \times M \times N \times w \times L/r/r/r)$ in the average case. ■

V. EXPERIMENTAL RESULTS AND PERFORMANCE EVALUATIONS

In this section, we evaluate performance of NOSEP on DNA sequences, time series data, protein sequence databases, and clickstream data from an e-commerce. We will report mining capacities in subsection V-C. In subsection V-D we will evaluate the performances of algorithms. In subsection V-E we will show the performances for different parameters. All experiments are conducted on a computer with an Intel Core I5, 3.4GHz CPU, 8.0GB DDR2 of RAM, Windows 7, and a 64 bit operating system. VC++ 6.0 is used to develop all algorithms, including GSgrow, NetM-B, NetM-D, NOSEP-B, and NOSEP, which can be downloaded from http://wuc.scse.hebut.edu.cn/nettree/nosep/.

A. Benchmark Datasets

Table II summarizes the 16 benchmark datasets used in our experiments. The datasets are mainly from three domains: 1) time series datasets; 2) DNA and protein amino acid sequences; and 3) clickstream data in e-commerce.

B. Baseline Methods

1) gd-DSPMiner [24]: This method carries out contrast sequence pattern mining under no-condition.

2) SAIL [38] and SBO [39]: They are two pattern matching strategies to approximately calculate the support under the one-off condition (which is an NP-hard problem).

3) GSgrow [28]: This method carries out sequence pattern mining under the nonoverlapping condition.

4) NOSEP-B: As we mentioned in challenge 1 in subsection IV-A, the backtracking strategy is less effective to calculate the support. To verify the analysis, we propose NOSEP-B which adopts the backtracking strategy and employs the same pruning strategy as NOSEP to reduce the candidate space.

5) NetM-B and NetM-D: These two algorithms employ NETGAP to calculate the support, but adopt the breadth-first search and the depth-first search which were employed by MAPB and MAPD in our previous research [16], respectively. NetM-B stores the frequent patterns in a queue while NetM-D stores the frequent patterns in a stack. The principle of NetM-B and NetM-D are shown as follows. We get a frequent pattern from the queue/ stack at first. According to the *Apriori* property, all its supper-patterns are candidate patterns. We calculate the supports of these supper-patterns and then find the frequent patterns. At last we enqueue or push the frequent patterns into the queue or stack and iterate the above process until the queue or stack is empty.

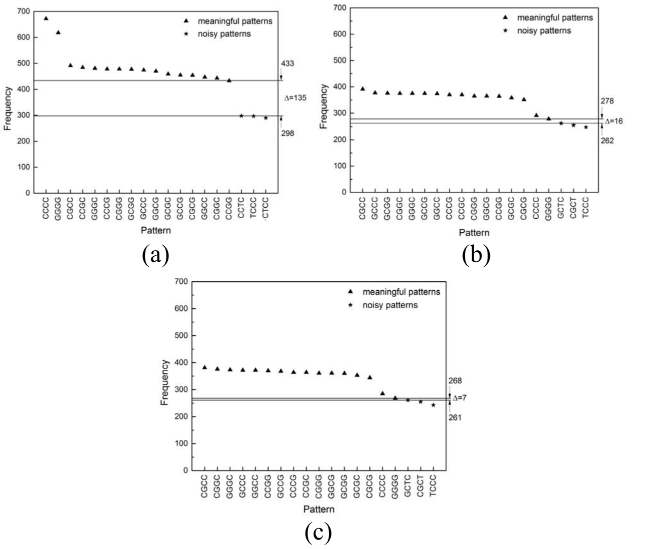


Fig. 3. Comparison of the use of the nonoverlapping condition versus the one-off condition for DNA sequence mining. The region between a triangle and a star denotes the margin between meaningful patterns and noisy patterns. The larger the margin, the better the algorithm is for discovering good patterns (i.e., less noise prone). (a) NOSEP. (b) SBO. (c) SAIL.

C. Sequence Pattern Validity Comparison

In previous sections, we have discussed that three types of conditions are commonly used for sequence pattern mining: 1) nonoverlapping condition; 2) no-condition; and 3) one-off condition. In order to demonstrate that patterns discovered using nonoverlapping condition are potentially more useful, we carry out experiments on DNA sequences and time series sequences to demonstrate the validity of discovered sequence patterns using different conditions.

1) Pattern Validity on DNA Sequences: gd-DSPMiner proposed in [24] mined the most patterns from dataset TSS which are composed by C and G. The result is consistent with the famous "CpG islands" observation. We also set the same gap constraints as [24] which are $len = [1, 15]$ and $gap = [1, 2]$. We mine the top 19 patterns with length 4. We report their frequencies in Fig. 3, where the points on the left of the triangle (inclusive) denote the meaningful CpG islands of DNA sequences, which are intervals of sequences that are of high frequency in C and G [44], while the points on the right of the star denote noisy patterns.

Fig. 3(a) shows that the minimum frequency of meaningful CpG islands patterns and the maximum frequency of noisy patterns under the nonoverlapping condition are 433 and 298, respectively. Hence, the gap under the nonoverlapping condition is $433 - 298 = 135$. Under the one-off condition using selecting better occurrence (SBO), the minimum and maximum frequencies are 278 and 262, respectively, according to Fig. 3(b). Under the one-off condition using string matching with wildcards and length constraints (SAIL), the minimum and maximum frequencies are 268 and 261, respectively, according to Fig. 3(c). The gaps under the one-off condition using SBO and SAIL are 16 and 7, respectively. Therefore, the rates of gap ratios under the nonoverlapping condition and under the one-off condition using SAIL are 135/433 and 7/268, respectively. Since [24] mined a noisy pattern "CCTC" we can safely conclude that NOS is easier to mine the useful patterns without noisy patterns than the other mining methods.

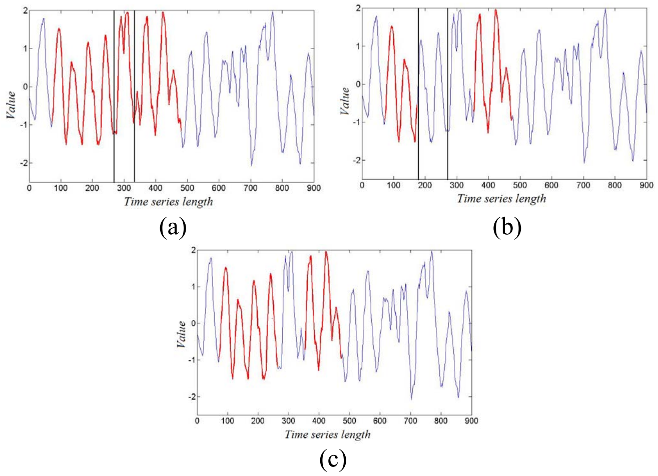


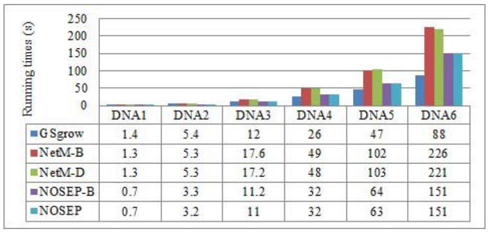
Fig. 4. Frequent patterns in the training set of WormsTwoClass. (a) No-condition. (b) One-off condition. (c) Nonoverlapping condition.

2) Pattern Validity on Time Series Sequence: In this section, wormstwoclass (WTC) is selected to demonstrate the capability of the discovered patterns for classification. Three kinds of mining methods: 1) no-condition; 2) the one-off condition; and 3) the nonoverlapping condition, are used to mine frequent patterns in the sequences and the parameters are $gap = [5, 15]$ and $len = [1, 150]$. All the methods can mine the frequent pattern "dcd." In order to show the difference in the mining results, the 46th piece of training time series in WTC is selected and the corresponding time series are shown in Fig. 4.

The red curve in Fig. 4 means a frequent pattern in time series data. Compared with Fig. 4(c), we can make the following observations. 1) A part of the red curve in Fig. 4(a) is apparently different from the others. It is a kind of over-expression and 2) A part of the curve in Fig. 4(b) is similar to other frequent patterns, but it fails to be mined. It is a kind of under-expression. The reasons are as follows. All three figures describe the pattern $P = p_1p_2p_3 =$ dcd using different methods. But under-expression occurs under the one-off condition, since it is the most restrictive one and requires that each position in the sequence only appears once in any of the occurrences. For example, there is only one occurrence for pattern "d[0,2]c[0,2]d" in sequence "dcdcd" under the one-off condition, either $\langle 1, 2, 3\rangle$ or $\langle 3, 4, 5\rangle$, while there are two occurrences under the nonoverlapping condition, which means that both $\langle 1, 2, 3\rangle$ and $\langle 3, 4, 5\rangle$ can be matched under the nonoverlapping condition. Therefore, under-expression can happen under the one-off condition. There are three occurrences for pattern d[0,2]c[0,2]d in sequence "dcdcdxd" under no-condition, $\langle 1, 2, 3\rangle$, $\langle 3, 4, 5\rangle$, and $\langle 3, 4, 7\rangle$, while there are two occurrences under the nonoverlapping condition, $\langle 1, 2, 3\rangle$ and $\langle 3, 4, 5\rangle$. Apparently, $\langle 3, 4, 7\rangle$ is an over-expression occurrence under no-condition. Hence, from this example, we know that mining under the nonoverlapping

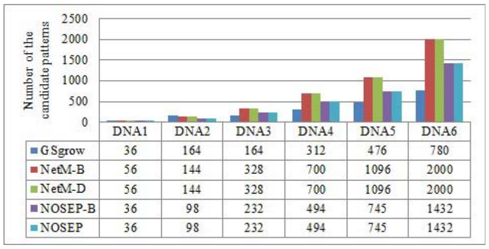
TABLE III
COMPARISON OF THE NUMBER OF MINED PATTERNS IN DNA SEQUENCES

	DNA1	DNA2	DNA3	DNA4	DNA5	DNA6
GSgrow	9	26	41	78	119	195
NetM-B	14	36	82	175	274	500
NetM-D	14	36	82	175	274	500
NOSEP-B	14	36	82	175	274	500
NOSEP	14	36	82	175	274	500



	DNA1	DNA2	DNA3	DNA4	DNA5	DNA6
■ GSgrow	1.4	5.4	12	26	47	88
■ NetM-B	1.3	5.3	17.6	49	102	226
■ NetM-D	1.3	5.3	17.2	48	103	221
■ NOSEP-B	0.7	3.3	11.2	32	64	151
■ NOSEP	0.7	3.2	11	32	63	151

Fig. 5. Comparison of the mining speed on DNA sequences.

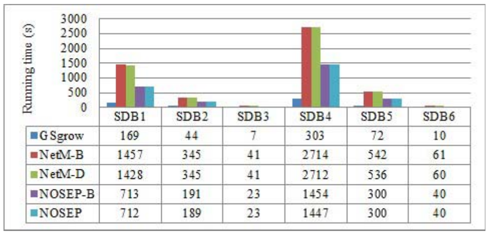


	DNA1	DNA2	DNA3	DNA4	DNA5	DNA6
■ GSgrow	36	164	164	312	476	780
■ NetM-B	56	144	328	700	1096	2000
■ NetM-D	56	144	328	700	1096	2000
■ NOSEP-B	36	98	232	494	745	1432
■ NOSEP	36	98	232	494	745	1432

Fig. 6. Comparison of the candidate patterns on DNA sequences.

TABLE IV
COMPARISON OF THE NUMBER OF MINED PATTERNS IN PROTEIN SEQUENCE DATABASES

	SDB1	SDB2	SDB3	SDB4	SDB5	SDB6
GSgrow	1072	410	110	1721	584	156
NetM-B	1248	472	120	1928	672	165
NetM-D	1248	472	120	1928	672	165
NOSEP-B	1248	472	120	1928	672	165
NOSEP	1248	472	120	1928	672	165



	SDB1	SDB2	SDB3	SDB4	SDB5	SDB6
■ GSgrow	169	44	7	303	72	10
■ NetM-B	1457	345	41	2714	542	61
■ NetM-D	1428	345	41	2712	536	60
■ NOSEP-B	713	191	23	1454	300	40
■ NOSEP	712	189	23	1447	300	40

Fig. 7. Comparison of the mining speed in protein sequence databases.

condition can avoid under-expression and over-expression effectively.

D. *Algorithm Performance Comparison*

1) Experiments on DNA Sequences: In this section, we report the performance of different pattern mining algorithms on DNA1 ∼ DNA6 sequences. We set $len = [1, 20]$, $gap = [0, 3]$, and $minsup = 800$, and report the results in Table III. Figs. 5 and 6 show the comparisons of mining speed and the number of candidate patterns, respectively. The results indicate following two major observations.

NetM-B, NetM-D, NOSEP-B, and NOSEP Have Better Performance Than GSgrow: According to Table III, we know that NetM-B, NetM-D, NOSEP-B, and NOSEP have the same mining results and can find more frequent patterns than GSgrow, especially for long sequences. For example, for DNA6, NetM-B, NetM-D, and NOSEP can find 500 frequent patterns while GSgrow only finds 195 frequent patterns. Since INSgrow may lose feasible occurrences, the support of a pattern is less than the actual value. Therefore, some frequent patterns cannot be found using GSgrow, which employs INSgrow to calculate the support. NetM-B, NetM-D, and NOSEP, however, can mine all the frequent patterns, since they use the algorithm NETGAP to calculate the support and we show that NETGAP is a complete algorithm. NOSEP-B employs NETGAP-back which adopts the backtracking strategy to iteratively obtain the nonoverlapping occurrences and is also a complete algorithm. Hence, NetM-B, NetM-D, NOSEP-B, and NOSEP can find more frequent patterns than GSgrow.

NOSEP is Faster Than NOSEP-B, NetM-B, and NetM-D but Slower Than GSgrow: Fig. 5 shows that GSgrow is faster than other four algorithms. However, because GSgrow cannot find all frequent patterns, we will not take GSgrow into consideration. NOSEP is faster than the other three algorithms. For example, NOSEP-B, NetM-B, and NetM-D take 64, 109 and 105 s, respectively, while NOSEP takes only 63 s in DNA5 according to Fig. 5. The reason is that both NetM-B and NetM-D check 1096 candidate patterns while NOSEP only checks 745 candidate patterns according to Fig. 6. Hence, NOSEP is considerably faster than NetM-B and NetM-D. NOSEP is a little bit faster than NOSEP-B since the backtracking strategy is occasionally used. For example, in Fig. 2, when NOSEP-B finds suboccurrence ⟨8, 10, 11⟩ and node n_4^{12} as the only child of node n_3^{11} has been used by occurrence ⟨5, 7, 9, 12⟩. Therefore, node n_4^{12} cannot be used by suboccurrence ⟨8, 10, 11⟩. Hence, the backtracking strategy is triggered and NOSEP-B finds occurrence ⟨8, 10, 14, 16⟩ at last. There are three nonoverlapping occurrences and the backtracking strategy is triggered only once. So the backtracking strategy is occasionally used in Example 5. Hence, NOSEP is a little bit faster than NOSEP-B.

2) Experiments on Protein Sequences: To further evaluate the performance of the mining algorithms, we select six databases with SBD1 ∼ SDB6 and set $len = [1, 30]$, $gap = [0, 5]$, and $minsup = 500$, and the mining results of GSgrow, NetM-B, NetM-D, NOSEP-B, and NOSEP are shown in Table IV. Figs. 7 and 8 show the comparisons of the mining speed and the number of candidate patterns, respectively.

The experimental results show that the five algorithms can be applied to not only a single sequence, but also a sequence database. According to Table IV, we know that NetM-B, NetM-D, NOSEP-B, and NOSEP have the same mining results and can find more frequent patterns than GSgrow, especially for long sequences. According to Fig. 7, we know that GSgrow is much faster than the other three algorithms. But GSgrow cannot find all the frequent patterns. NOSEP is faster than

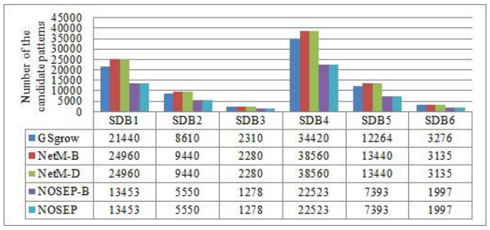


Fig. 8. Comparison of the number of candidate patterns in protein sequence databases.

TABLE V
COMPARISON OF THE MINING SPEED ON GAZELLE DATABASES (S)

	BMS1 $minsup$ = 700	BMS1 $minsup$ = 800	BMS1 $minsup$ = 900	BMS2 $minsup$ = 700	BMS2 $minsup$ = 800	BMS2 $minsup$ = 700
GSgrow	93	76	70	1266	933	716
NetM-B	451	340	309	54490	24990	24644
NetM-D	191	105	81	1055	497	335
NOSEP-B	184	99	75	809	387	288
NOSEP	157	85	65	712	343	254

TABLE VI
COMPARISON OF THE NUMBER OF CANDIDATE PATTERNS ON GAZELLE DATABASES

	BMS1 $minsup$ = 700	BMS1 $minsup$ = 800	BMS1 $minsup$ = 900	BMS2 $minsup$ = 700	BMS2 $minsup$ = 800	BMS2 $minsup$ = 700
GSgrow	29559	21042	18036	374080	250500	193720
NetM-B	3127	1638	1224	8400	3900	2610
NetM-D	3127	1638	1224	8400	3900	2610
NOSEP-B	2810	1521	1156	5694	2737	2038
NOSEP	2810	1521	1156	5694	2737	2038

NOSEP-B, NetM-B, and NetM-D. The reason for this is that NOSEP can effectively reduce the candidate patterns according to Fig. 8. For example, NetM-B and NetM-D take 1597 and 1589 s, respectively, while NOSEP only takes 712 s in SDB1 according to Fig. 7. These phenomena can also be found in DNA sequences. The reason for this is that both NetM-B and NetM-D check 26 229 candidate patterns while NOSEP only checks 13 494 candidate patterns. Hence, NOSEP is much - D. From Figs. 7 and 8 we can see that a lower number of candidate patterns will require a lower runtime. For example, NOSEP calculates 1278 and 13494 candidate patterns and runs 23 and 712 s in SDB3 and SDB1, respectively. The same phenomenon can also be found in DNA sequences.

3) Experiments on Gazelle Datasets: To further evaluate the performance on large alphabet size, we set *gap* = [0, 200] and *len* = [1, 200] for BMS1 and BMS2. We also set *minsup*=700, *minsup*=800, and *minsup*=900. All mining algorithms can find the same patterns in the same instances and the number of mined patterns for these instances are 59, 42, 36, 112, 75, and 58, respectively. The mining speed and the number of candidate patterns are shown in Tables V and VI, respectively.

GSgrow has the same mining results as NOSEP. The reason is shown as follows. We know that GSgrow cannot find occurrence $\langle 8, 10, 14, 16\rangle$ based on suboccurrence $\langle 8, 10, 11\rangle$ in Example 5 since GSgrow does not employ the backtracking strategy. The missing feasible occurrence phenomenon may occur when there are multichoices in the same gap constraint. However, in large alphabet size dataset, there is less chance for multichoice to occur. The risk of missing feasible occurrence goes down when the alphabet size increases. This phenomenon can also be found in DNA and protein experiments. We know that the alphabet sizes of DNA and protein are 4 and 20, respectively. According to Tables III and IV, GSgrow finds 195 patterns while NOSEP finds 500 patterns in DNA6. Over 60% patterns are lost in the DNA experiment. However, GSgrow finds 1722 patterns while NOSEP finds 1929 patterns in SDB4. About 10% patterns are lost in the protein experiment. Therefore, GSgrow obtains the same results as NOSEP for Gazelle datasets.

However, we ought to stress that we can see that GSgrow runs slower than NOSEP in many cases especially in BMS2 according to Table V. The reason lies that BMS2 contains 3340 distinct items while BMS1 contains 497. As we know that GSgrow employs the depth-first search to mine frequent patterns. According to Example 8, we show that pattern growth approach is more effective than the depth-first search. From Table VI, we see that GSgrow checks 1 93 720 candidate patterns while NOSEP only checks 2038 candidate patterns in BMS2 with *minsup* = 900. This means that NOSEP only checks about 1% patterns of that of GSgrow in this instance. Similar, NOSEP only checks about 6% patterns of GSgrow in BMS1 with *minsup* = 900. This phenomenon can also be seen in DNA and protein experiments. For example, GSgrow checks 36 162 candidate patterns while NOSEP does 22 564 in SDB4. So NOSEP checks about 60% patterns of GSgrow in protein instance. Meanwhile, GSgrow checks 780 candidate patterns while NOSEP does 1432 in DNA6. NOSEP checks about two times of that of GSgrow in DNA. Hence, with the increase of the alphabet the running time of GSgrow increases which means that the performance of GSgrow goes down remarkable. Therefore, experimental results show that NOSEP adopts better pruning strategy than GSgrow.

To summarize, when the alphabet size is small, NOSEP runs slow, but can find more frequent patterns in the same case. When the alphabet size is large, the mining results are the same, but NOSEP runs fast. All in all, NOSEP has better performance than the state-of-the-art algorithms.

E. Performance Evaluations With Respect to Different Parameters

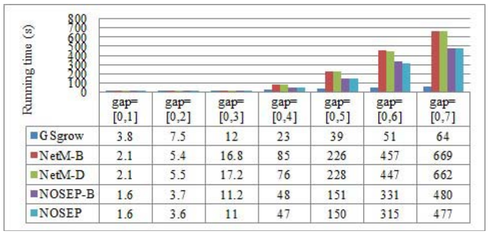
1) Length of Sequences: Obviously, the experiments in the above sections show the relationship between sequence length and the number of mined patterns, mining speed, and the number of candidate patterns. When the sequence length increases, the number of mined patterns also increases and the mining time will also increase rapidly. For example, the lengths of DNA1 and DNA6 are 6000 and 16 000, respectively. The number of frequent patterns increases from 14 to 500, and the mining time increases correspondingly from 0.7 to 151 s with the NOSEP algorithm. Other algorithms also show similar growth. The reason for this is that when the length of sequence increases, the support of a pattern will increase, more patterns

TABLE VII
COMPARISON OF THE NUMBER OF MINED PATTERNS UNDER DIFFERENT GAPS IN DNA SEQUENCES

	gap =[0,1]	gap =[0,2]	gap =[0,3]	gap =[0,4]	gap =[0,5]	gap =[0,6]	gap =[0,7]
GSgrow	16	29	41	82	138	178	227
NetM-B	16	34	82	275	650	1117	1446
NetM-D	16	34	82	275	650	1117	1446
NOSEP-B	16	34	82	275	650	1117	1446
NOSEP	16	34	82	275	650	1117	1446

TABLE VIII
COMPARISON OF THE NUMBER OF MINED PATTERNS UNDER DIFFERENT GAPS IN PROTEIN DATABASES

	gap =[0,1]	gap =[0,2]	gap =[0,3]	gap =[0,4]	gap =[0,5]	gap =[0,6]	gap =[0,7]
GSgrow	106	202	261	355	584	1063	1678
NetM-B	108	203	265	385	672	1264	2132
NetM-D	108	203	265	385	672	1264	2132
NOSEP-B	108	203	265	385	672	1264	2132
NOSEP	108	203	265	385	672	1264	2132



	gap=[0,1]	gap=[0,2]	gap=[0,3]	gap=[0,4]	gap=[0,5]	gap=[0,6]	gap=[0,7]
■ GSgrow	3.8	7.5	12	23	39	51	64
■ NetM-B	2.1	5.4	16.8	85	226	457	669
■ NetM-D	2.1	5.5	17.2	76	228	447	662
■ NOSEP-B	1.6	3.7	11.2	48	151	331	480
■ NOSEP	1.6	3.6	11	47	150	315	477

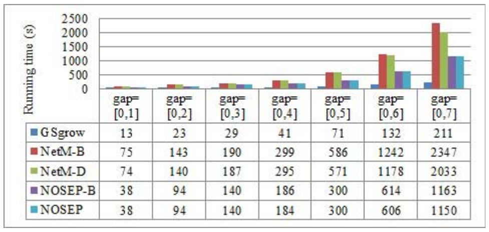
Fig. 9. Comparison of the mining speed under different gaps in DNA sequences (s).

can be frequent patterns, and it takes more time to calculate the support. Therefore, the mining time increases rapidly when sequence length increases.

2) Mining Threshold: Obviously, the experiments on Gazelle databases show the relationship between the threshold and the number of mined patterns and mining speed. The experimental results show that the higher *minsup* is, the fewer mined patterns there tend to be and the faster the mining speed is. For example, in BMS2 experiments, when *minsup* increases from 700 to 900, the number of mined patterns and the mining time decrease from 112 to 58 and from 712 to 254 s, respectively, with NOSEP according to Table V. The reason lies that the mining method satisfies the *Apriori* property. Therefore, when *minsup* increases, fewer patterns will be frequent patterns. Then the number of candidate pattern tends to be smaller. Hence, the mining time decreases while *minsup* increases.

3) Gap Constraints: In the rest of the experiments, DNA3 and SDB5 are selected. In order to evaluate the relationships between the gap constraints and the number of mined patterns and mining speed, in the DNA experiments, we set $len = [1, 20]$ and $minsup = 800$ and in the protein experiments, we set $len = [1, 30]$ and $minsup = 500$. The results are shown in Tables VII and VIII and Figs. 9 and 10.

The experimental results show that the larger gap is, the larger the number of mined patterns tends to be and the slower the mining speed is. For example, in the DNA experiments, when gap increases from [0, 1] to [0, 7], the number of mined patterns and mining time increase from 16 to 1446 and from 1.6 to 477 s, respectively, with NOSEP according to Table VII and Fig. 9. The protein experiments also exhibit similar phenomena. It can be easily understood that when gap increases, more patterns will be frequent patterns. Then the number of candidate patterns increases. Hence, the mining time increases when gap increases.



	gap=[0,1]	gap=[0,2]	gap=[0,3]	gap=[0,4]	gap=[0,5]	gap=[0,6]	gap=[0,7]
■ GSgrow	13	23	29	41	71	132	211
■ NetM-B	75	143	190	299	586	1242	2347
■ NetM-D	74	140	187	295	571	1178	2033
■ NOSEP-B	38	94	140	186	300	614	1163
■ NOSEP	38	94	140	184	300	606	1150

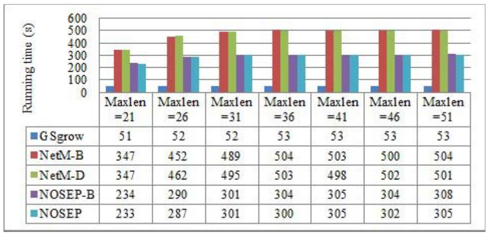
Fig. 10. Comparison of the mining speed under different gaps in protein databases (s).

TABLE IX
COMPARISON OF THE NUMBER OF MINED PATTERNS UNDER DIFFERENT LENGTHS IN DNA SEQUENCES

	$maxlen$ =21	$maxlen = 26$	$maxlen = 31$	$maxlen = 36$	$maxlen = 41$	$maxlen = 46$	$maxlen = 51$
GSgrow	231	236	238	239	239	239	239
NetM-B	1116	1388	1460	1479	1481	1481	1481
NetM-D	1116	1388	1460	1479	1481	1481	1481
NOSEP-B	1116	1388	1460	1479	1481	1481	1481
NOSEP	1116	1388	1460	1479	1481	1481	1481

TABLE X
COMPARISON OF THE NUMBER OF MINED PATTERNS UNDER DIFFERENT LENGTHS IN PROTEIN DATABASES

	$maxlen$ =5	$maxlen = 7$	$maxlen = 9$	$maxlen = 11$	$maxlen = 13$	$maxlen = 15$	$maxlen = 17$
GSgrow	255	305	371	505	584	584	584
NetM-B	258	308	422	584	672	672	672
NetM-D	258	308	422	584	672	672	672
NOSEP-B	258	308	422	584	672	672	672
NOSEP	258	308	422	584	672	672	672

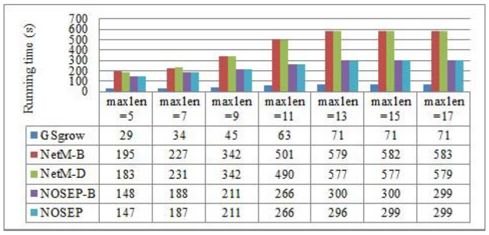


	Maxlen =21	Maxlen =26	Maxlen =31	Maxlen =36	Maxlen =41	Maxlen =46	Maxlen =51
■ GSgrow	51	52	52	53	53	53	53
■ NetM-B	347	452	489	504	503	500	504
■ NetM-D	347	462	495	503	498	502	501
■ NOSEP-B	234	290	301	304	305	304	308
■ NOSEP	233	287	301	300	305	302	305

Fig. 11. Comparison of the mining speed under different lengths in DNA sequences (s).

4) Length Constraint: To evaluate the relationships between the maximum length constraint and the number of mined patterns and mining speed, in the DNA experiments, we select $gap = [0, 4]$, $minlen = 1$, and $minsup = 600$ and in the protein experiments, we set $gap = [0, 5]$, $minlen = 1$, and $minsup = 500$. The results are shown in Tables IX and X and Figs. 11 and 12.

Obviously, when the maximum length constraint is less than the feasible maximum length, the number of patterns meeting the constraint and mining time will increase with the increase



	maxlen =5	maxlen =7	maxlen =9	maxlen =11	maxlen =13	maxlen =15	maxlen =17
■ GSgrow	29	34	45	63	71	71	71
■ NetM-B	195	227	342	501	579	582	583
■ NetM-D	183	231	342	490	577	577	579
■ NOSEP-B	148	188	211	266	300	300	299
■ NOSEP	147	187	211	266	296	299	299

Fig. 12. Comparison of the mining speed under different lengths in protein databases (s).

in *maxlen*. On the other hand, when the maximum length constraint exceeds the feasible maximum length, the number of mined patterns and mining time will not increase with the increase in maxlen. For example, in the protein experiments, when minsup is 500, the length of the longest frequent patterns is 3 in SDB5, and the maximum feasible length is $3 + (3 - 1) \times 5 = 13$ with the $gap = [0, 5]$. It can be seen from Table X that when maxlen exceeds 13, no matter how much *maxlen* increases, the results are the same for all mining algorithms.

According to Fig. 12, the running time is a little bit different when *maxlen* exceeds 13, and all five algorithms share similar phenomena. This is mainly attributed to the fact that the actual running time may vary even though running the same instances twice. When maxlen is less than 13, the number of mined patterns and mining time increase correspondingly with increases in *maxlen*. For example, $maxlen = 9$, $maxlen = 13$, and $maxlen = 15$, the number of mined patterns with NOSEP is 422, 672, and 672, respectively. Similar phenomena are shown in DNA experiments. When $minsup = 600$, the longest length of frequent patterns is 9. In $gap = [0, 4]$, the maximum feasible length is $9 + (9 - 1) \times 4 = 41$. So before reaching 41, with the increase in *maxlen*, the number of mined patterns and mining time increase accordingly. But after reaching 41, no matter how much maxlen increases, the number of mined patterns and the mining time remain unchanged. The same results can be observed in Table IX.

VI. Conclusion

Sequence pattern mining with gap constraints is inherently difficult to tackle, mainly because of difficulties in counting the pattern occurrences and in reducing the candidate pattern space. For all existing sequence pattern mining methods, counting of pattern occurrences is mainly based on three approaches: 1) no-condition; 2) the one-off condition; and 3) the nonoverlapping condition. All existing methods are either anti-*Apriori* or incomplete, and patterns discovered by these methods are either too restrictive or too general but cannot represent meaningful knowledge underneath the sequences. In this paper, we focus on an NOSEP mining task with gap constraints, where an NOSEP allows sequence letters to be utilized flexibly for pattern discovery. An effective mining algorithm, NOSEP, is proposed to solve NOS. NOSEP not only meets the *Apriori* property but is also a complete algorithm. It employs an effective algorithm to completely calculate the support and also adopts an effective pattern growth approach to effectively reduce the candidate patterns. Experimental results in DNA sequence mining and time series data mining demonstrate that NOSEP can discover more frequent patterns than state-of-the-art algorithms under the same conditions.

Further research can be carried in following directions.

1) In this paper, the support is used to find the frequent patterns. However, there are many other methods to determine the frequent patterns, such as frequency, occupancy [4], etc which can also be investigated.
2) More important applications will be conducted. In our opinion, the ways to effectively mine patterns as features for sequence classification [9], [10], as one of the important applications, which are frequent in the positive class and infrequent in the negative class should be considered in the future.
3) Pattern mining without involving user-specified gaps could be explored.

Acknowledgment

The authors would like to thank three anonymous reviewers for their valuable comments and suggestions.

References

[1] C. C. Aggarwal and J. Han, *Frequent Pattern Mining*. Cham, Switzerland: Springer, 2014.

[2] S. Ventura and J. M. Luna, *Pattern Mining With Evolutionary Algorithms*. Cham, Switzerland: Springer, 2016.

[3] C. Li, Q. Yang, J. Wang, and M. Li, "Efficient mining of gap-constrained subsequences and its various applications," *ACM Trans. Knowl. Disc. Data*, vol. 6, no. 1, p. 2, 2012.

[4] B. Le, M.-T. Tran, and B. Vo, "Mining frequent closed inter-sequence patterns efficiently using dynamic bit vectors," *Appl. Intell.*, vol. 43, no. 1, pp. 74–84, 2015.

[5] S. Zhang, Z. Du, and J. T. L. Wang, "New techniques for mining frequent patterns in unordered trees," *IEEE Trans. Cybern.*, vol. 45, no. 6, pp. 1113–1125, Jun. 2015.

[6] L. Zhang *et al.*, "Occupancy-based frequent pattern mining*," *ACM Trans. Knowl. Disc. Data*, vol. 10, no. 2, p. 14, 2015.

[7] F. Min, Y. Wu, and X. Wu, "The Apriori property of sequence pattern mining with wildcard gaps," *Int. J. Funct. Informat. Personalised Med.*, vol. 4, no. 1, pp. 15–31, 2012.

[8] C. D. Tan, F. Min, M. Wang, H.-R. Zhang, and Z.-H. Zhang, "Discovering patterns with weak-wildcard gaps," *IEEE Access*, vol. 4, pp. 4922–4932, 2016.

[9] F. Rasheed and R. Alhajj, "A framework for periodic outlier pattern detection in time-series sequences," *IEEE Trans. Cybern.*, vol. 44, no. 5, pp. 569–582, May 2014.

[10] H. Jiang, J. Zhang, H. Ma, N. Nazar, and Z. Ren, "Mining authorship characteristics in bug repositories," *Sci. China Inf. Sci.*, vol. 60, no. 1, pp. 1–16, 2017.

[11] E. Egho, D. Gay, M. Boullé, N. Voisine, and F. Clérot, "A user parameter-free approach for mining robust sequential classification rules," *Knowl. Inf. Syst.*, vol. 52, no. 1, pp. 53–81, 2017.

[12] C. Zhou, B. Cule, and B. Goethals, "Pattern based sequence classification," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 5, pp. 1285–1298, May 2016.

[13] X. Wu, X. Zhu, Y. He, and A. N. Arslan, "PMBC: Pattern mining from biological sequences with wildcard constraints," *Comput. Biol. Med.*, vol. 43, no. 5, pp. 481–492, 2013.

[14] J. Ge, Y. Xia, J. Wang, C. H. Nadungodage, and S. Prabhakar, "Sequential pattern mining in databases with temporal uncertainty," *Knowl. Inf. Syst.*, vol. 51, no. 3, pp. 821–850, 2017.

[15] H. Yang *et al.*, "Mining top-k distinguishing sequential patterns with gap constraint," *J. Softw.*, vol. 26, no. 11, pp. 2994–3009, 2015.

[16] Y. Wu, L. Wang, J. Ren, W. Ding, and X. Wu, "Mining sequential patterns with periodic wildcard gaps," *Appl. Intell.*, vol. 41, no. 1, pp. 99–116, 2014.

[17] M. Zhang, B. Kao, D. W. Cheung, and K. Y. Yip, "Mining periodic patterns with gap requirement from sequences," *ACM Trans. Knowl. Disc. Data*, vol. 1, no. 2, p. 7, 2007.

[18] H.-F. Wang *et al.*, "Efficient mining of distinguishing sequential patterns without a predefined gap constraint," *Chin. J. Comput.*, vol. 39, no. 10, pp. 1979–1991, 2016.

[19] P. Bille, I. L. Gørtz, H. W. Vildhøj, and D. K. Wind, "String matching with variable length gaps," *Theor. Comput. Sci.*, vol. 443, pp. 25–34, Jul. 2012

[20] X. Wu, J.-P. Qiang, and F. Xie, "Pattern matching with flexible wildcards," *J. Comput. Sci. Technol.*, vol. 29, no. 5, pp. 740–750, 2014.

[21] D. Guo, X. Hu, F. Xie, and X. Wu, "Pattern matching with wildcards and gap-length constraints based on a centrality-degree graph," *Appl. Intell.*, vol. 39, no. 1, pp. 57–74, 2013.

[22] Y. Wu, C. Shen, H. Jiang, and X. Wu, "Strict pattern matching under non-overlapping condition," *Sci. China Inf. Sci.*, vol. 60, no. 1, pp. 1–16, 2017.

[23] S. Ghosh, M. Feng, H. T. Nguyen, and J. Li, "Risk prediction for acute hypotensive patients by using gap constrained sequential contrast patterns," in *Proc. AMIA Annu. Symp. Amer. Med. Inf. Assoc.*, Washington, DC, USA, 2014, p. 1748.

[24] X. Wang, L. Duan, G. Dong, Z. Yu, and C. Tang, "Efficient mining of density-aware distinguishing sequential patterns with gap constraints," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2014, pp. 372–387.

[25] S.-J. Yen and Y.-S. Lee, "Mining non-redundant time-gap sequential patterns," *Appl. Intell.*, vol. 39, no. 4, pp. 727–738, 2013.

[26] X. Zhu and X. Wu, "Mining complex patterns across sequences with gap requirements," in *Proc. Int. Joint Conf. Artif. Intell.*, Hyderabad, India, 2007, pp. 2934–2940.

[27] H. T. Lam, F. Mörchen, D. Fradkin, and T. Calders, "Mining compressing sequential patterns," *Stat. Anal. Data Min.*, vol. 7, no. 1, pp. 34–52, 2013.

[28] B. Ding, D. Lo, J. Han, and S.-C. Khoo, "Efficient mining of closed repetitive gapped subsequences from a sequence database," in *Proc. IEEE Int. Conf. Data Eng.*, Shanghai, China, 2009, pp. 1024–1035.

[29] Y. Feng *et al.*, "Mining spatial-temporal patterns and structural sparsity for human motion data denoising," *IEEE Trans. Cybern.*, vol. 45, no. 12, pp. 2693–2706, Dec. 2015.

[30] L. Hui, Y-C. Chen, J. T.-Y. Weng, and S.-Y. Lee, "Incremental mining of temporal patterns in interval-based database," *Knowl. Inf. Syst.*, vol. 46, no. 2, pp. 423–448, 2016.

[31] L. Nie, H. Jiang, Z. Ren, Z. Sun, and X. Li, "Query expansion based on crowd knowledge for code search," *IEEE Trans. Services Comput.*, vol. 9, no. 5, pp. 771–783, Sep./Oct. 2016.

[32] H. Jiang *et al.*, "ROSF: Leveraging information retrieval and supervised learning for recommending code snippets," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2016.2592909.

[33] Y. Wu, X. Wu, F. Min, and Y. Li, "A Nettree for pattern matching with flexible wildcard constraints," in *Proc. IEEE Int. Conf. Inf. Reuse Integr.*, Las Vegas, NV, USA, 2010, pp. 109–114.

[34] Y. Wu, Z. Tang, H. Jiang, and X. Wu, "Approximate pattern matching with gap constraints," *J. Inf. Sci.*, vol. 42, no. 5, pp. 639–658, 2016.

[35] Y. Wu, S. Fu, H. Jiang, and X. Wu, "Strict approximate pattern matching with general gaps," *Appl. Intell.*, vol. 42, no. 3, pp. 566–580, 2015.

[36] M. K. Warmuth and D. Haussler, "On the complexity of iterated shuffle," *J. Comput. Syst. Sci.*, vol. 28, no. 3, pp. 345–358, 1984.

[37] X. Chai, X.-F. Jia, Y.-X. Wu, H. Jiang, and X.-D. Wu, "Strict pattern matching with general gaps and one-off condition," *J. Softw.*, vol. 26, no. 5, pp. 1096–1112, 2015.

[38] G. Chen, X. Wu, X. Zhu, A. N. Arslan, and Y. He, "Efficient string matching with wildcards and length constraints," *Knowl. Inf. Syst.*, vol. 10, no. 4, pp. 399–419, 2006.

[39] Y. Wu, X. Wu, H. Jiang, and F. Min, "A heuristic algorithm for MPMGOOC," *Chin. J. Comput.*, vol. 34, no. 8, pp. 1452–1462, 2011.

[40] Y. Chen *et al.*. *The UCR Time Series Classification Archive*. Accessed: Jul. 2015. [Online]. Available: www.cs.ucr.edu/~eamonn/time_series_data/

[41] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proc. ACM SIGMOD Workshop Res. Issues Data Min. Knowl. Disc.*, San Diego, CA, USA, 2003, pp. 2–11.

[42] T. Wittkop, J. Baumbach, F. P. Lobo, and S. Rahmann, "Large scale clustering of protein sequences with FORCE—A layout based heuristic for weighted cluster editing," *BMC Bioinformat.*, vol. 8, no. 1, p. 396, 2007.

[43] P. Fournier-Viger *et al.*, "SPMF: A Java open-source pattern mining library," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3389–3393, 2014.

[44] C. Bock *et al.*, "CpG island methylation in human lymphocytes is highly correlated with DNA sequence, repeats, and predicted DNA structure," *PLoS Genet.*, vol. 2, no. 3, pp. 243–252, 2006.

Youxi Wu received the Ph.D. degree in theory and new technology of electrical engineering from the Hebei University of Technology, Tianjin, China.

He is currently a Ph.D. Supervisor and a Professor with the Hebei University of Technology. His current research interests include data mining and machine learning.

Yao Tong received the master's degree in computer science and technology from the Hebei University of Technology, Tianjin, China.

Her current research interest includes data mining.

Xingquan Zhu (SM'12) received the Ph.D. degree in computer science from Fudan University, Shanghai, China.

He is an Associate Professor with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL, USA, and a Distinguished Visiting Professor (Eastern Scholar) with the Shanghai Institutions of Higher Learning, Shanghai. His current research interests include data mining, machine learning, and multimedia systems.

Dr. Zhu is an Associate Editor of the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, from 2008 to 2012, and since 2014.

Xindong Wu (F'11) received the Ph.D. degree from the University of Edinburgh, Edinburgh, U.K.

He is a Yangtze River Scholar with the Hefei University of Technology, Hefei, China, and a Professor of computer science with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA, USA. His current research interests include data mining, big data analytics, knowledge based systems, and Web information exploration.

Dr. Wu is the Steering Committee Chair of the IEEE International Conference on Data Mining and the Editor-in-Chief of *Knowledge and Information Systems*. He is a Fellow of the American Association for the Advancement of Science.