NOSEP: Non-Overlapping Sequence Pattern Mining with Gap Constraints

Youxi Wu, Yao Tong, Xingquan Zhu, *Senior, 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. However, 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 since the support or support ratio of a pattern may be greater than that of its sub-patterns. 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 non-overlapping sequence pattern mining task with gap constraints, where a non-overlapping sequence pattern allows sequence letters to be flexible, yet maximally, utilized for pattern discovery. A new non-overlapping sequence pattern mining algorithm (NOSEP), which is an Apriori-based and complete mining algorithm, is proposed by using Nettree, a specially designed data structure, to calculate the exact occurrence of a pattern in the sequence. Experimental results and comparisons with biology DNA sequences, time series data, and Gazelle Datasets demonstrate the efficiency of the proposed algorithms and the uniqueness of non-overlapping sequence patterns compared to other methods.**

I. Introduction

Sequence pattern mining aims to discover frequent subsequences as patterns in a sequence or a sequence database [1]. Such patterns are strongly correlated to meaningful events or knowledge within the data and are therefore commonly applied to numerous fields, such as mining customer purchase patterns [2], mining tree-structure information [3], travel-landscape recommendations [4], time-series analysis and prediction [5], [6], [7], bug repositories [8], sequence classification [9], [10], and biological sequence data analysis [11], [12]. Because a sequence pattern consists of multiple pattern letters occurring in a sequence order, it is possible that when pattern occurring in the sequence, two pattern letters may appear in the required sequence order but with different numbers of letters between them. Formally, a frequent sequence pattern with gap constraints (or flexible wildcards or wildcard gaps) is defined as $P = p_1[min_1, max_1]p_2...[min_{m-1}, max_{m-1}]p_m$[13], [14], [15], [16], [17], [18], [19], [20]. If $min_1 = min_2 = ... = min_{m-1} = a$, and $max_1 = max_2 = ... = max_{m-1} = b$, it can be called pattern with periodic gap constraints (or periodic wildcard gaps) [14], [16] and P can be written as $p_1p_2...p_m$ with $gap = [a, b]$. According to [16], pattern P=C[0,2]G[1,3]C is not a pattern with periodic gap constraints, 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, therefore 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 [21], mining biological characteristics [22], mining customer purchase patterns [23], feature extraction [1], 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 this challenge, the state-of-the-art mining methods [1], [11], [14], [16], [24], [25], [26] mainly rely on three different conditions to count the pattern frequency: no condition [1], [14], [16], [24], the one-off condition [11], [25], and the non-overlapping condition [26]. Compared with the one-off condition and the non-overlapping condition, some researches do not employ any other condition, to make it easy to understand. In this paper, we therefore 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 non-overlapping condition allows a sequence letter to match and rematch pattern letter as long as the matched patterns letters are different (i.e., non-overlapping). Therefore, the non-overlapping condition is less restrictive than the one-off condition and more specific than

Y. Wu and Y. Tong are with School of Computer Science and Engineering, Hebei University of Technology, Tianjin 300401, China, School of Economics and Management, Hebei University of Technology, Tianjin 300130, China, and Hebei Province Key Laboratory of Big Data Calculation, Tianjin 300401, China (e-mail:wuc@scse.hebut.edu.cn)

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

X. Wu is with School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, Anhui, China and School of Computing and Informatics, University of Louisiana at Lafayette, Louisiana 70504, USA (e-mail:xwu@louisiana.edu)

This research is supported by the National Natural Foundation of China under grant Nos.61673159 and 61370144, and the Graduate Student Innovation Program of Hebei Province under grant No. 220056.

no condition for pattern mining. As a result, it is possible to find more meaningful patterns from the sequences.

In addition to the above benefits, to the best of our knowledge, existing sequence pattern mining with gaps cannot achieve a 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 [11], [25] or under the non-overlapping condition [26]. On the other hand, some methods can calculate the support of the patterns exactly, but both the support and support ratio of a super-pattern can be greater than its sub-pattern, that is, 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 [14], [16], [24].

The above observations motivated the proposed research which intends to use the non-overlapping condition for mining sequence patterns with gap constraints. Compared to existing sequence pattern mining algorithms, the proposed 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 contributions of the paper are threefold: (1) we propose a new algorithm, NETGAP using Nettree data structure, to compute the support of a candidate pattern in the sequence and prove the completeness of the algorithm; (2) we propose NOSEP, a new algorithm employing a pattern growth strategy to reduce the candidate space for effective sequence pattern mining with gap constraints; and (3) experiments on DNA sequence mining and time series data mining demonstrate that NOSEP can discover more frequent patterns than state-of-the-art algorithm under the same conditions, and patterns discovered by NOSEP are more effective than those discovered by other competitive algorithms.

The remainder of the paper is organized as follows. Related work is introduced in Section II, followed by the problem definition in Section III. Section IV proposes a pattern matching algorithm NETGAP to compute the exact support of a pattern, including the proof of completeness, and the NOSEP algorithm for sequence pattern mining. Section V reports the algorithm performance and comparisons, and we conclude the paper in Section VI.

II. Related work

Sequence pattern mining has been widely applied in various fields [27], [28], [29], [30], [21]. 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: no condition [1], [16], [14], [24], the one-off condition [11], [25] and the non-overlapping condition [26]. Here, we point out that compared with other conditions, a kind of methods without adding any condition, therefore, we call no condition in this paper. An illustrative example is shown as follows to clarify the relationships of these methods.

	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

Example 1. *Suppose we have sequence $S = s_1s_2s_3s_4s_5 = CGCGC$ and 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 are shown in Figure 1.*

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 [11], [25] and the non-overlapping condition [26]. 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 non-overlapping condition as described in Ref. [26]), each of the characters in the sequence can be used only once. But under the non-overlapping condition, s_j cannot be reused by the same p_i and can be reused by other p_i [26]. 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 non-overlapping occurrences under the non-overlapping condition, since $\langle 3 \rangle$ matches p_3 and p_1, respectively.

Since pattern matching strategy can be used to calculate the support in the sequence pattern mining task, therefore, pattern matching methods are also considered in a comparison of related work shown in Table I.

Now, we briefly review the contributions of these related works. Many researchers focused on sequence pattern mining under no condition. For example, Zhang et al [16] firstly addressed sequence pattern mining with periodic gap constraints and proposed MPP-Best algorithm to tackle the problem. Zhu and Wu [24] proposed a more effective algorithm GCS which can be used to deal with a set of sequences. Our previous research [14] employed Nettree data structure and proposed MAPD algorithm which is more effective than the previous the state-of-the-art algorithms. Li et al [14] 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

TABLE I: A COMPARISON OF RELATED WORK

	Type of research	Type of condition	Periodic gap constraints	Length constraints	Support	Mining type	Pruning strategy
Li et al [1]	Pattern mining	No condition	Yes	No	Exact	Exact	Apriori-like
Zhang et al [16]			Yes	No			
Zhu and Wu [24]			Yes	No			
Wu et al [14]			Yes	No			
Bille et al [17]	Pattern matching	Other/No condition [1]	No	No	Exact	–[2]	–
Wu et al [31]		No condition	No	Yes	Exact	–	–
Wu et al [11]	Pattern mining	One-off condition	No	No	Approximate	Approximate	Apriori
Lam et al [25]			No	No			
Guo et al [19]	Pattern matching	One-off condition	No	Yes	Approximate	–	–
Ding et al [26]	Pattern mining	Non-overlapping condition	Yes	Yes	Approximate	Approximate	Apriori
Wu et al [20]	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 [17] 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.

under no condition lies that under no condition the support of P in S can be calculated exactly according to many previous researches [16], [24], [14]. Many researches also focused on pattern matching under no condition. These researches are beyond the limitations of periodic gap constraints [17], [31], exact pattern matching [32], and positive gap constraints[33]. 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 sub-pattern 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 [5], [6] , 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 non-overlapping condition is that these two methods meet the Apriori property, since it is easy to see that the support of a pattern is no greater than that of its sub-pattern. Many researchers focused on pattern matching under the one-off condition. For example, without user-specified gap constraints, Wu et al [11] employed the one-off condition and proposed an effective algorithm PMBC to discover pattern from biological sequences. Lam et al [25] proposed two minimum description length based algorithms for mining non-redundant 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 [34]. Although many heuristic algorithms [18], [19], [35], [36], [37] 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 [26] firstly addressed the sequence pattern mining under the non-overlapping 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 non-overlapping 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 I in subsection IV-A NETGAP. Therefore, GSgrow and CloGSgrow are approximate mining algorithms. From Table I, we can see that besides [26] another most relevant research is one of our previous works [20] which deals with a pattern matching issue. In that research, we proved that the problem of calculating the support of a pattern under the non-overlapping condition is in P and proposed a complete algorithm named NETLAP-Best which iteratively finds the rightmost non-overlapping occurrences to calculate the support. This paper however handles a sequence pattern mining issue. We will propose NOSEP algorithm to mine all frequent patterns, and NOSEP employs another complete algorithm NETGAP which iteratively finds the leftmost non-overlapping occurrence to calculate the support and a pattern growth strategy to reduce the space of candidate patterns. Our previous work [14] also mined the frequent sequence patterns, but according to Table I, it is easy to see that it focused on mining the patterns under no condition while this paper deals with the patterns under the non-overlapping condition. Therefore, the two works focus on different issues. Moreover, in [14], we employed the last level of a Nettree, named Incomplete Nettree structure, to calculate the support of a pattern. And then, the Incomplete Nettree was used to create other Incomplete Nettrees of all its super-patterns and calculated their supports in a one-way scan. Therefore, the data structures are different. Furthermore, we proposed two algorithms MAPB (employed the breadth-first search) and MAPD (employed the depth-first search) to find the frequent patterns under no condition. Hence, the pruning strategies are also different. Therefore, this study differs from the previous studies.

To the best of our knowledge, existing sequence pattern mining with gap constraints cannot achieve a balance between the Apriori property and completeness. In this paper, we investigate non-overlapping sequence pattern mining with gap constraints, with the aim of discovering the complete set of frequent sequence patterns based on the Apriori property.

III. Problem definition

In this section, we define important definitions and use examples to illustrate the problem.

Definition 1. *(Pattern with gap constraints and sequence) Pattern P with gap constraints can be described as $p_1[min_1, max_1]p_2...[min_{m-1}, max_{m-1}]p_m$, where $p_j \in \Sigma$, min_j and max_j are two non-negative integers and represent the minimum gap constraint and maximum gap constraint, respectively, $1 \leq j \leq m$, 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 $p_1[a, b]p_2...[a, b]p_m$ or $P = p_1p_2...p_m$ with $gap = [a, b]$, where $0 \leq a \leq b$. Sequence S with length n can be written as $s_1s_2...s_i...s_n$, where $1 \leq i \leq n$ 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 $L = \langle l_1, l_2, ..., l_m\rangle$ is called an occurrence of P in S, if and only if $1 \leq l_1 < l_2 < ... < l_m \leq n$, $min_j \leq l_{j+1} - l_j - 1 \leq max_j$, $p_1 = s_{l_1}$,$p_2 = s_{l_2}$,..., 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 $L = \langle l_1, l_2, ...l_m\rangle$ satisfies $minlen \leq l_m - l_1 + 1 \leq maxlen$, 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 $L = \langle l_1, l_2, ..., l_m\rangle$ and $L' = \langle l'_1, l'_2, ..., l'_m\rangle$ be two occurrences. If and only if $\forall 1 \leq j \leq m$: $l_j \neq l'_j$, L and L' are two non-overlapping occurrences. If any two occurrences in a set are non-overlapping, then the set is called non-overlapping occurrence set. The support of P in S under the non-overlapping condition, which is denoted by $sup(P, S)$, is the size of the maximum non-overlapping occurrence set.*

As we know that all occurrences of P with gap constraints $gap = [0, 2]$ and length constraint $len = [1, 5]$ in S in Example 2 is $\langle 1, 2, 3\rangle$, $\langle 1, 2, 5\rangle$, $\langle 1, 4, 5\rangle$, and $\langle 3, 4, 5\rangle$. We can see that the maximum non-overlapping occurrence set is $\{\langle 1, 2, 3\rangle, \langle 3, 4, 5\rangle\}$ in this example. Therefore, under the non-overlapping condition, $sup(P, S)$ is 2. In particular, we do not consider the relationship between an occurrence and its sub-occurrence. 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, $SDB = \{S_1, S_2, ..., S_N\}$,where N is the size of sequence database. The support of pattern P in SDB is the sum of supports of P in $S_1, S_2, ..., S_N$, respectively, 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 non-overlapping sequence pattern mining with gap constraints) 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 non-overlapping sequence pattern mining with gap constraints 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. Algorithms

For sequence pattern mining tasks, there are two main factors that affect the mining performance: calculation of the support and reduction of the candidate pattern space. Accordingly, in Section IV-A we propose an effective algorithm, named NETGAP, to calculate the support and we prove the completeness of the algorithm. Section IV-B proposes the mining algorithm to reduce the candidate pattern space. Moreover, we show the space and time complexities in Section IV-C and briefly introduce the principles of three competitive algorithms in Section IV-D.

A. NETGAP

There are two major challenges in calculating the support: (1) effectively checking a 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 sub-pattern 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, it is easy to see that the first non-overlapping occurrence of sub-pattern A[0,1]T is $\langle 1, 2\rangle$. But there is no non-overlapping 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 non-overlapping occurrence of super-pattern P = A[0,1]T[0,1]C. INSgrow [26] calculates the occurrences for a pattern based on the sub-occurrences for its sub-pattern. 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 non-overlapping 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 non-overlapping 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 [31] 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 j-th 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 non-root node n_j^i ($j > 1$) may have multiple parents $\{n_{j-1}^{i_1}, n_{j-1}^{i_2}, ..., n_{j-1}^{i_m}\}$ ($m \geq 1$), and thus there may be multiple paths from a node to a root node.

Definition 8. *(Absolute leaf) A leaf in the m-th 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 [31] 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. □

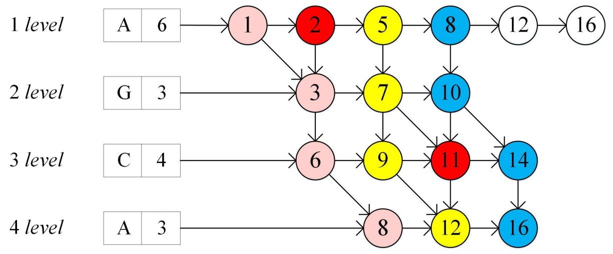
Definition 10. *(The minimal full path, the minimal occurrence, the maximal full path, and the maximal occurrence) A full path that iterates the leftmost child from the min-root to its leaf is called the minimal full path. The corresponding occurrence of the minimal full path is called the minimum occurrence. 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 non-overlapping occurrences.*

Proof. A and B are $\langle n_1^{a_1}, n_2^{a_2}, ..., n_m^{a_m}\rangle$ and $\langle n_1^{b_1}, n_2^{b_2}, ..., n_m^{b_m}\rangle$, 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, $\langle a_1, a_2, ..., a_m\rangle$ and $\langle b_1, b_2, ..., b_m\rangle$ are two non-overlapping 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 some concepts of Definitions 8-10.

Example 5. *Suppose we have sequence $S = s_1s_2s_3s_4s_5s_6s_7s_8s_9s_{10}s_{11}s_{12}s_{13}s_{14}s_{15}s_{16} = AAGTACGACGCATCTA$ and pattern $P = A[0,3]G[0,3]C[0,3]A$, a Nettree shown as follows can be created according to the algorithm CreateNettree in previous work[20]. 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.*



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: The corresponding Nettree

It is easy for us to obtain the first minimal occurrence $\langle 1,3,6,8\rangle$, marked in pink in Figure 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 non-overlapping occurrences of P in S; that is, $sup(P,S) = 3$.

Lemma 3. *If a node has no child, then the node can be pruned.*

Proof. If n_j^i is a non-absolute leaf node in a Nettree, then 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.

Algorithm 1 NETGAP

```
Input: Sequence $S$, Pattern $P$, $gap = [a,b]$, $len = [minlen, maxlen]$, and $minsup$
Output: $sup(P,S)$
 1: Create a $nettree$ of $P$ in $S$;
 2: Prune nodes which have no child according to Lemma 3;
 3: for each $n_1^i$ in $nettree$ do
 4:     $node[1] \leftarrow n_1^i$; //node used to store an occurrence;
 5:     for $j$=1 to $nettree.level - 1$ step 1 do
 6:         $node[j+1]$ is the leftmost child that meets the length constraints of $node[j]$;
 7:     end for
 8:     $sup(P,S)$++;
 9:     if $sup(P,S) > minsup$ then return $sup(P,S)$;
10:     Prune nodes which have no child according to Lemma 3;
11: end for
12: return $sup(P,S)$;
```

Lemma 4. *Let $\langle x_j, x_{j+1}\rangle$ and $\langle y_j, y_{j+1}\rangle$ be two sub-occurrences of sub-pattern $p_j[a_j,b_j]p_{j+1}$. 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 sub-occurrences.*

Proof. Our previous work [20] has shown that if $\langle x_j, x_{j+1}\rangle$ and $\langle y_j, y_{j+1}\rangle$ are two sub-occurrences, then $\langle x_j, y_{j+1}\rangle$ and $\langle y_j, x_{j+1}\rangle$ are also two sub-occurrences, where $x_j < y_j$ and $x_{j+1} > y_{j+1}$. □

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 sub-occurrences of sub-pattern T[0,2]C. According to Lemma 4, we can safely say $\langle 3,5\rangle$ and $\langle 4,6\rangle$ are two sub-occurrences.*

Theorem 1. *The algorithm NETGAP is complete.*

Proof. The proof is given in Appendix A. □

B. NOSEP

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

Definition 11. *(Prefix and suffix sub-pattern and super-pattern) Suppose we have sequence $P = p_1p_2...p_m$ and event items a and b. If $Q = Pa$, then P is called the prefix sub-pattern of Q and is denoted by $prefix(Q) = P$. Q is a super-pattern of P. Similarly, if $R = bP$, then P is called the suffix sub-pattern of R and denoted by $suffix(R) = P$ and $R = bP$. Since $prefix(Q) = suffix(R) = P$, 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 sub-pattern and the suffix sub-pattern of P are ACC and CCT, respectively. If $Q =$ CCTG, then $T = P \oplus Q =$ ACCTG.*

Theorem 2. *The non-overlapping sequence pattern mining with gap constraints satisfies the Apriori property.*

Proof. The proof is given in Appendix A. □

Our previous work [14] 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, MAPB and MAPD [14] 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 non-overlapping 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.

Example 8. *We find all frequent patterns from sequence $S = s_1s_2s_3s_4s_5s_6s_7s_8s_9s_{10}s_{11}s_{12}s_{13}s_{14}s_{15}s_{16}$ =AAGTACGACG CATCTA, $minsup = 3$, 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 it is easy for us to know that there are seven kinds of frequent patterns with length 2; that is, {'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', '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.*

Firstly, 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 pattern 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.

To mine the frequent patterns quickly, when we detect the support of a pattern is greater than $minsup$, we should stop calculating the support as early as possible since this process is slow. NOSEP is shown in Algorithm 2.

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: $len = 1$;
 3: $C =$gen_candidate($meta[len]$); // call gen_candidate algorithm, put the results in
    candidate set $C$
 4: while $C <> null$ do
 5:     for each $cand$ in $C$ do
 6:         $minsup1 = minsup$;
 7:         for each sequence $s_k$ in $SDB$ do
 8:             $sup1 =$ NETGAP($cand$, $minsup1$);
 9:             $minsup1 -= sup1$;
10:             $sup(cand) += sup1$;
11:             if $sup(cand) \geq minsup$ then
12:                 $meta[len + 1]$.enqueue($cand$);
13:                 break;
14:             end if
15:         end for
16:     end for
17:     $len = len + 1$;
18:     $C =$gen_candidate($meta[len]$);
19: end while
20: return $meta[1] \cup meta[2] \ldots \cup meta[len]$;
```

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

Algorithm 3 gen_candidate(m)

```
Input: $m$
Output: Candidate set $C$
 1: $start = 1$;
 2: for $i = 1$ to $|m|$ do
 3:     $R = suffix(m[i])$;
 4:     $Q = prefix(m[start])$;
 5:     if $R <> Q$ then
 6:         $start =$ binarysearch($m$, $Q$, 1, $|m|$);
 7:     end if
 8:     if $start >= 1$ && $start <= |m|$ then
 9:         while $Q == R$ do
10:             $C$.enqueue($R \oplus Q$);
11:             $start = start + 1$;
12:             if $start > |m|$ then
13:                 $start = 1$;
14:                 $R = prefix(m[start])$;
15:             end if
16:         end while
17:     end if
18: end for
19: return $C$;
```

C. Complexities

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

Proof. The proof is given in Appendix A. □

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 the average case, where m, w, and r are given in the space complexity, N and L are the length of SDB and the number of the candidate patterns, respectively.*

Proof. The proof is given in Appendix A. □

V. Experimental results and performance evaluations

In this section, we validate the mining capacities, the performance and scalability of NOSEP on DNA sequences, time series data, protein sequence databases and clickstream data from an e-commerce. In particular, we will report the comparison of the mining capacities in Section V.C. In Section V. D we will evaluate the performances of algorithms. In Section 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, NetMining-B, NetMining-D, NOSEP-Back, and NOSEP, which can be downloaded from http://wuc.scse.hebut.edu.cn/nettree/nosep/.

A. Benchmark Datasets

Table II shows datasets used in this paper for clarification.

B. Baseline Methods

1) gd-DSPMiner [22]: This method is a kind of contrast sequence pattern mining under no condition.

2) SAIL [36] and SBO [37]: These methods are two pattern matching strategies to approximately calculate the support under the one-off condition since the issue is an NP-hard problem.

3) GSgrow [26]: This method is a kind of sequence pattern mining under the non-overlapping condition.

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

5) NetMining-B and NetMining-D: These two algorithms also employ the algorithm 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 [14], respectively. NetMining-B stores the frequent patterns in a queue while NetMining-D stores the frequent patterns in a stack. The principle of NetMining-B and NetMining-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.

TABLE II: DATABASES

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 [22], 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 [38] (http://www.cs.ucr.edu/~eamonn/time_series_data/) is converted to demonstrate the capability of the discovered patterns for classification using SAX [39] (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 [40].
Note 5: Gazelle datasets, provided by Fournier-Viger P et al [41], had been used by [26] and [42].

C. Comparison of mining capacities

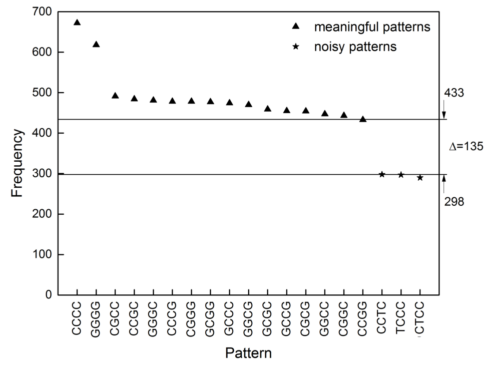
As we know besides the non-overlapping condition, there are no condition and the one-off condition which are introduced in this paper. We briefly review the relationships and characteristics of the three conditions in related work section. In this subsection, two kinds of experiments, pattern discovery from DNA sequences and the discovered patterns for classification in time series sequence, are employed to illustrate the difference of the mining capacities.

1) The mining capacity in DNA sequences: Pattern discovery from DNA sequences for gene transcription site discovery is a defined biological sequence mining task, because a gene transcription site is a special subsequence carrying meaningful biological functions. To discover the top 19 patterns with length 4 from dataset TS, we set $len = [1, 15]$ and $gap = [1, 2]$ which is the same gap constraints as in Ref. [22]. We report their frequencies in Figure 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 high frequency in C and G [43], and the points on the right of a star denote noisy patterns.

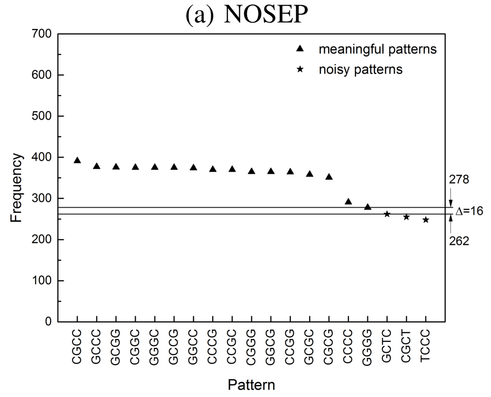
From Figure 3 (a) we can see that the minimum frequency of meaningful "CpG islands" patterns and the maximum frequency of noisy patterns under the non-overlapping condition are 433 and 298, respectively. Hence, the gap under the non-overlapping condition is 433-298=135. Under the one-off condition using SBO, the minimum and maximum frequencies are 278 and 262, respectively, according to Figure 3 (b). Under the one-off condition using SAIL, the minimum and maximum frequencies are 268 and 261, respectively, according to Figure 3 (c). Therefore, the gaps under the one-off condition using SBO and SAIL are 16 and 7, respectively. Therefore, the rates of gap ratios under the non-overlapping condition and under the one-off condition using SAIL are 135/433 and 7/268, respectively. Since Ref. [22] also mined a noisy pattern CCTC, we can safely conclude that if sequence pattern mining with periodic gap constraints is suitable for CpG island discovery, patterns under the non-overlapping condition is easier to mine the useful patterns without noisy patterns than the other conditions.

2) The mining capacity in a time series sequence: In this subsection, WTC is selected to demonstrate the capability of the discovered patterns for classification. Three kinds of mining methods, no condition, the one-off condition, and the non-overlapping 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 46^{th} piece of training time series in WTC is selected and the corresponding time series are shown in Figure 4.

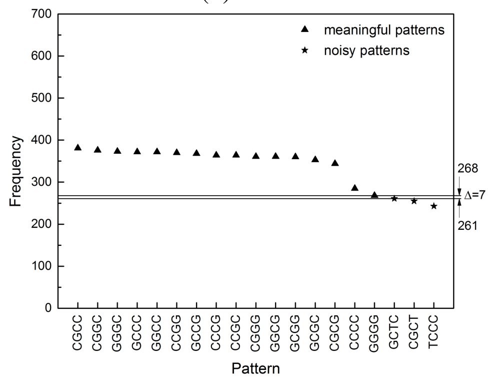
The red curve in Figure 4 means a frequent pattern in time series data. Compared with Figure 4 (c), we can make the following observations: (1) A part of the red curve in Figure 4 (a) is apparently different from the others. It is a kind of over-expression. (2) A part of the curve in Figure 4 (b) is 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 non-overlapping condition, which



(a) NOSEP



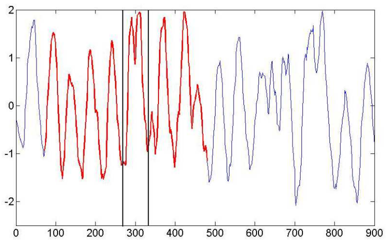
(b) SBO



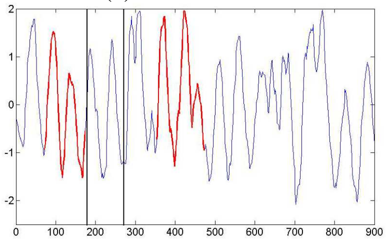
(c) SAIL

Fig. 3: Comparison of the use of the non-overlapping 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).

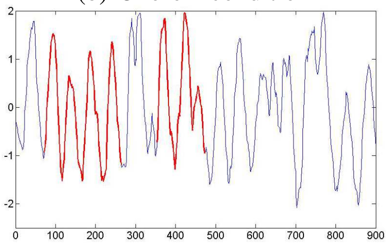
means that both $\langle 1,2,3\rangle$ and $\langle 3,4,5\rangle$ can be matched under the non-overlapping condition. Therefore, under-expression can happen under the one-off condition. There are three occurrences for pattern 'd[0,2]c[0,2]d' in sequence 'dcdcdxd' under no condition, $\langle 1,2,3\rangle$, $\langle 3,4,5\rangle$, and $\langle 3,4,7\rangle$, while there are 2 occurrences under the non-overlapping 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 non-overlapping condition can avoid under-expression and over-expression effectively.



(a) No condition



(b) One-off condition



(c) Non-overlapping condition

Fig. 4: The frequent patterns in the training set of WormsTwoClass

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
NetMining-B	14	36	82	175	274	500
NetMining-D	14	36	82	175	274	500
NOSEP-Back	14	36	82	175	274	500
NOSEP	14	36	82	175	274	500

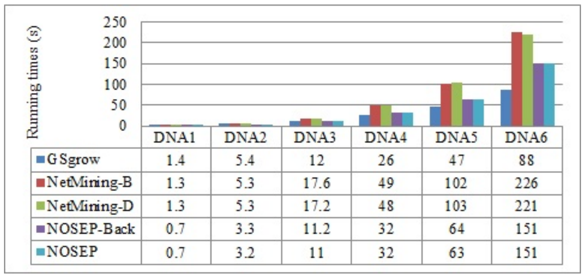
D. Comparison of the performances of algorithms

1) Experimental results on DNA sequences: In order to verify the performances of the mining algorithms, DNA1 $\sim$ DNA6 are selected in this subsection. We set $len = [1, 20]$, $gap = [0, 3]$, and $minsup = 800$, and the mining results are shown in Table II. Figure 5 and Figure 6 show the comparisons of mining speed and the number of candidate patterns, respectively.

1. NetMining-B, NetMining-D, NOSEP-Back, and NOSEP have better performance than GSgrow.

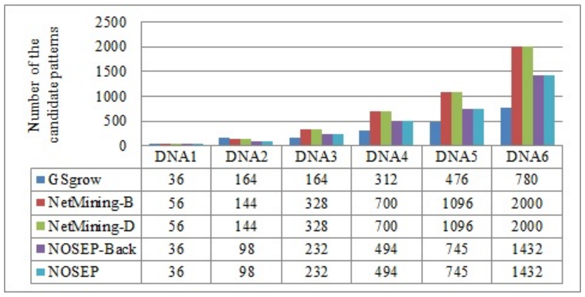
According to Table III, we know that NetMining-B, NetMining-D, NOSEP-Back, and NOSEP have the same mining results and can find more frequent patterns than GSgrow, especially for long sequences. For example, for DNA6, NetMining-B, NetMining-D, and NOSEP can find 500 frequent patterns while GSgrow only 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 IN-

Sgrow to calculate the support. NetMining-B, NetMining-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-Back employs NETGAP-Back which adopts the backtracking strategy to iteratively obtain the non-overlapping occurrences and is also a complete algorithm. Hence, NetMining-B, NetMining-D, NOSEP-Back, and NOSEP can find more frequent patterns than GSgrow.



	DNA1	DNA2	DNA3	DNA4	DNA5	DNA6
GSgrow	1.4	5.4	12	26	47	88
NetMining-B	1.3	5.3	17.6	49	102	226
NetMining-D	1.3	5.3	17.2	48	103	221
NOSEP-Back	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 in DNA sequences



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

Fig. 6: Comparison of the candidate patterns in DNA sequences

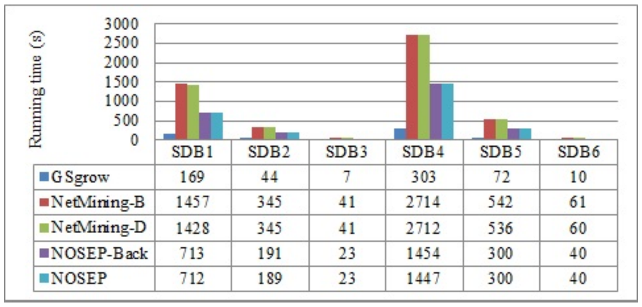
2. NOSEP is faster than NOSEP-Back, NetMining-B, and NetMining-D but slower than GSgrow.

Figure 5 reports that GSgrow is faster than the other four algorithms. But GSgrow cannot find all the frequent patterns. So we do not take GSgrow into consideration. NOSEP is faster than the other three algorithms. For example, NOSEP-Back, NetMining-B, and NetMining-D take 64, 109 and 105 s, respectively, while NOSEP takes only 63 s in DNA5 according to Figure 5. The reason is that both NetMining-B and NetMining-D check 1096 candidate patterns while NOSEP only checks 745 candidate patterns according to Figure 6. Hence, NOSEP is considerably faster than NetMining-B and NetMining-D. NOSEP is a little bit faster than NOSEP-Back since the backtracking strategy is occasionally used. For example, in Fig. 2, when NOSEP-Back finds sub-occurrence $\langle 8, 10, 11\rangle$ and node n_4^{12} as the only child of node n_3^{11} has been used by occurrence $\langle 5, 7, 9, 12\rangle$. Therefore, node n_4^{12} cannot be used by sub-occurrence $\langle 8, 10, 11\rangle$. Hence, the backtracking strategy is triggered and NOSEP-Back finds occurrence $\langle 8, 10, 14, 16\rangle$ at last. There are 3 non-overlapping 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-Back.

TABLE IV: Comparison of the number of mined patterns in protein sequence databases

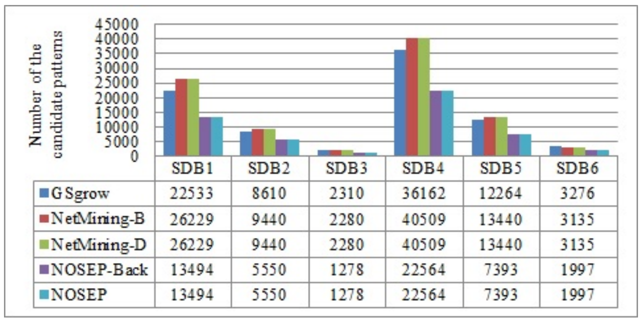
	SDB1	SDB2	SDB3	SDB4	SDB5	SDB6
GSgrow	1073	410	110	1722	584	156
NetMining-B	1249	472	120	1929	672	165
NetMining-D	1249	472	120	1929	672	165
NOSEP-Back	1249	472	120	1929	672	165
NOSEP	1249	472	120	1929	672	165

2) Experimental results for protein sequence databases: To further evaluate the performance of the mining algorithms, we select six databases with SBD1 $\sim$ SDB6 and set $len = [1, 30]$, $gap = [0, 5]$, and $minsup = 500$, and the mining results of GSgrow, NetMining-B, NetMining-D, NOSEP-Back, and NOSEP are shown in Table IV. Figure 7 and Figure 8 show the comparisons of the mining speed and the number of candidate patterns, respectively.



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

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



	SDB1	SDB2	SDB3	SDB4	SDB5	SDB6
GSgrow	22533	8610	2310	36162	12264	3276
NetMining-B	26229	9440	2280	40509	13440	3135
NetMining-D	26229	9440	2280	40509	13440	3135
NOSEP-Back	13494	5550	1278	22564	7393	1997
NOSEP	13494	5550	1278	22564	7393	1997

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

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 NetMining-B, NetMining-D, NOSEP-Back, and NOSEP have the same mining results and can find more frequent patterns than GSgrow, especially for long sequences. According to Figure 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 NOSEP-Back, NetMining-B, and NetMining-D. The reason for this is that NOSEP can effectively reduce the candidate patterns according to Figure 8. For example, NetMining-B and NetMining-D take 1597 and 1589 s, respectively, while NOSEP only takes 712 s in SDB1

TABLE V: Comparison of the number of mined patterns in Gazelle databases

	B1 700	B1 800	B1 900	B2 700	B2 800	B2 900
GSgrow	59	42	36	112	75	58
NOSEP	59	42	36	112	75	58

TABLE VI: Comparison of the mining speed in Gazelle databases

	B1 700	B1 800	B1 900	B2 700	B2 800	B2 900
GSgrow	93	76	70	1266	933	716
NOSEP	157	85	65	712	343	254

according to Figure 7. These phenomena can also be found in DNA sequences. The reason for this is that both NetMining-B and NetMining-D check 26229 candidate patterns while NOSEP only checks 13494 candidate patterns. Hence, NOSEP is much faster than NetMining-B and NetMining-D.

3) Experimental results for Gazelle datasets: To further evaluate the performance on large alphabet size, we set $gap = [0, 200]$ and $len = [1, 200]$ for BMS1 and BMS2. Due to the limitation of the space, we only show the performances of GSgrow and NOSEP. The number of mined patterns, the mining speed, and the number of candidate patterns are shown in Tables V, VI, and VII, respectively. In the tables, Bx Y refers the name of the dataset with $minsup$. For example, B1 800 means BMS1 with $minsup$ =800.

From Table V, we can see that GSgrow has the same mining results as NOSEP. The reason is shown as follows. It is easy to know that GSgrow cannot find occurrence $\langle 8, 10, 14, 16\rangle$ based on sub-occurrence $\langle 8, 10, 11\rangle$ in Example 5 since GSgrow does not employ the backtracking strategy. The missing feasible occurrence phenomenon may occur when there are multi-choices in the same gap constraint. However, in large alphabet size dataset, there is less chance for multi-choice 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 II 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 VI. 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 VII: Comparison of the number of candidate patterns in Gazelle databases

	B1 700	B1 800	B1 900	B2 700	B2 800	B2 900
GSgrow	29559	21042	18036	374080	250500	193720
NOSEP	2810	1521	1156	5694	2737	2038

TABLE VIII: Comparison of the number of mined patterns under different thresholds in DNA sequences

	$minsup$ =450	$minsup$ =500	$minsup$ =550	$minsup$ =600	$minsup$ =650	$minsup$ =700
GSgrow	262	196	145	106	89	71
NetMining-B	793	507	359	247	188	142
NetMining-D	793	507	359	247	188	142
NOSEP-Back	793	507	359	247	188	142
NOSEP	793	507	359	247	188	142

Table VII, we see that GSgrow checks 193720 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 36162 candidate patterns while NOSEP does 22564 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 2 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.

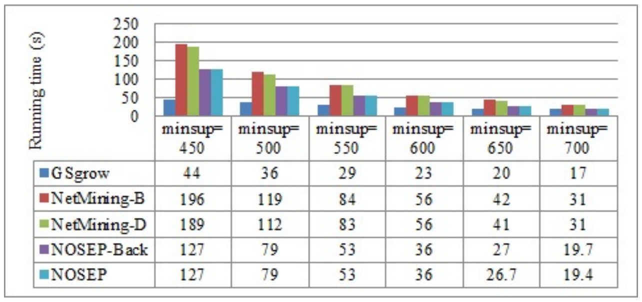
To summarize, when the alphabet size is small, NOSEP runs slow, but can find more frequent patterns in the same case. When the alphabet size is large, the mining results are the same, but NOSEP runs fast. All in all, NOSEP has better performance than the state-of-the-art algorithms.

E. Performance evaluations for different parameters

1) Length of sequence evaluation: Obviously, the experiments in the above subsections 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 16000, 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 can be frequent patterns, and it takes more time to calculate the support. Therefore, the mining time increases rapidly when sequence length increases.

2) Threshold evaluation: In the rest of the experiments, DNA3 and SDB5 are selected. To evaluate the relationships between the threshold and the number of mined patterns and mining speed, in the DNA experiments, we choose $len = [1, 20]$ and $gap = [0, 3]$ and in the protein experiments, we set $len = [1, 30]$ and $gap = [0, 5]$. The results are shown in Tables VIII and IX and Figures 9 and 10.

The experimental results show that the higher $minsup$ is, the fewer mined patterns there tend to be and the faster the



	minsup=450	minsup=500	minsup=550	minsup=600	minsup=650	minsup=700
🟦 GSgrow	44	36	29	23	20	17
🟥 NetMining-B	196	119	84	56	42	31
🟩 NetMining-D	189	112	83	56	41	31
🟪 NOSEP-Back	127	79	53	36	27	19.7
🟦 NOSEP	127	79	53	36	26.7	19.4

Fig. 9: Comparison of mined speed under different thresholds in DNA sequences

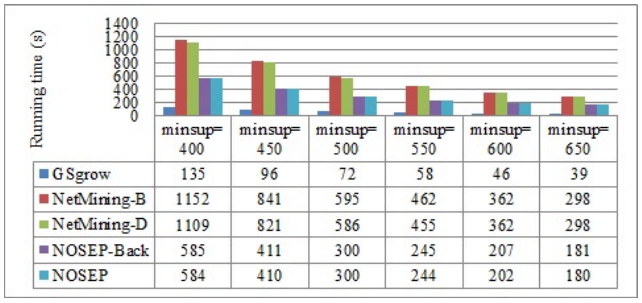
TABLE IX: Comparison of the number of the mining patterns under different thresholds in protein sequence databases

	$minsup$ =400	$minsup$ =450	$minsup$ =500	$minsup$ =550	$minsup$ =600	$minsup$ =650
GSgrow	1128	777	584	471	383	325
NetMining-B	1282	929	672	528	431	361
NetMining-D	1282	929	672	528	431	361
NOSEP-Back	1282	929	672	528	431	361
NOSEP	1282	929	672	528	431	361

mining speed is. For example, in the DNA experiments, when $minsup$ increases from 450 to 700, the number of mined patterns and the mining time decrease from 793 to 142 and from 127 to 19.4 s, respectively, with NOSEP according to Table VIII and Figure 9. The protein experiments also exhibit the same phenomena. It can be easily understood that when $minsup$ increases, fewer patterns will be frequent patterns. Then the number of candidate pattern tends to be smaller. Hence, the mining time decreases when $minsup$ increases.

3) Gap constraints evaluation: 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 X and XI and Figures 11 and 12.

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 477s, respectively, with NOSEP according to

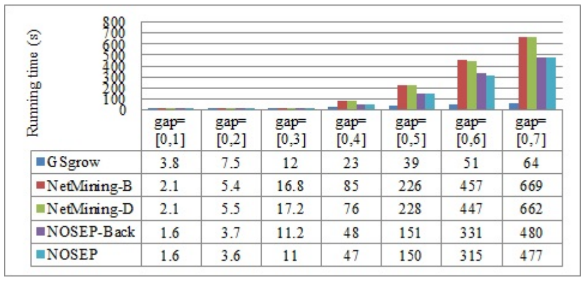


	minsup=400	minsup=450	minsup=500	minsup=550	minsup=600	minsup=650
🟦 GSgrow	135	96	72	58	46	39
🟥 NetMining-B	1152	841	595	462	362	298
🟩 NetMining-D	1109	821	586	455	362	298
🟪 NOSEP-Back	585	411	300	245	207	181
🟦 NOSEP	584	410	300	244	202	180

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

TABLE X: 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
NetMining-B	16	34	82	275	650	1117	1446
NetMining-D	16	34	82	275	650	1117	1446
NOSEP-Back	16	34	82	275	650	1117	1446
NOSEP	16	34	82	275	650	1117	1446



	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
🟥 NetMining-B	2.1	5.4	16.8	85	226	457	669
🟩 NetMining-D	2.1	5.5	17.2	76	228	447	662
🟪 NOSEP-Back	1.6	3.7	11.2	48	151	331	480
🟦 NOSEP	1.6	3.6	11	47	150	315	477

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

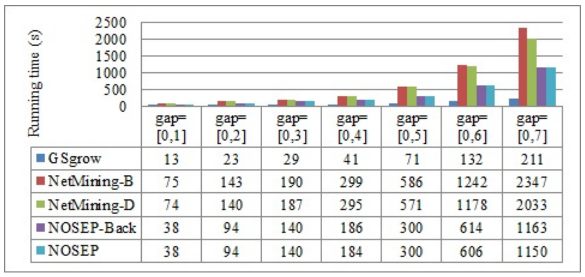
Table X and Figure 11. The protein experiments also exhibit similar phenomena. It can be easily understood that when gap increases, more patterns will be frequent patterns. Then the number of candidate patterns increases. Hence, the mining time increases when gap increases.

4) Length constraint evaluation: 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 XII and XIII and Figures 13 and 14.

Obviously, when the maximum length constraint is less than the feasible maximum length, the number of patterns

TABLE XI: 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
NetMining-B	108	203	265	385	672	1264	2132
NetMining-D	108	203	265	385	672	1264	2132
NOSEP-Back	108	203	265	385	672	1264	2132
NOSEP	108	203	265	385	672	1264	2132

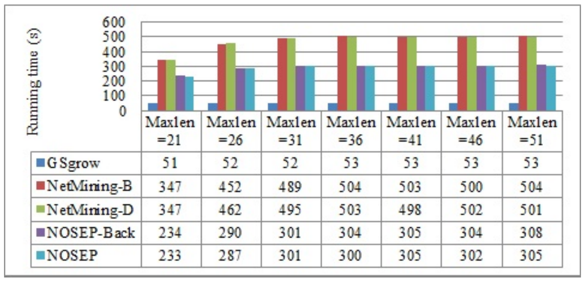


	gap=[0,1]	gap=[0,2]	gap=[0,3]	gap=[0,4]	gap=[0,5]	gap=[0,6]	gap=[0,7]
🟦 GSgrow	13	23	29	41	71	132	211
🟥 NetMining-B	75	143	190	299	586	1242	2347
🟩 NetMining-D	74	140	187	295	571	1178	2033
🟪 NOSEP-Back	38	94	140	186	300	614	1163
🟦 NOSEP	38	94	140	184	300	606	1150

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

TABLE XII: 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
NetMining-B	1116	1388	1460	1479	1481	1481	1481
NetMining-D	1116	1388	1460	1479	1481	1481	1481
NOSEP-Back	1116	1388	1460	1479	1481	1481	1481
NOSEP	1116	1388	1460	1479	1481	1481	1481



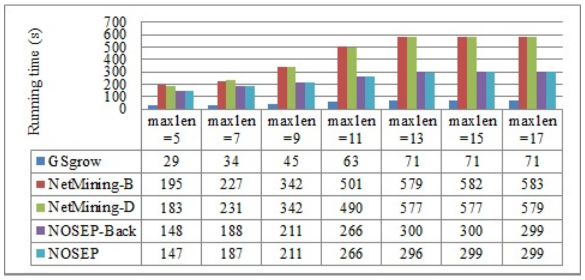
	Maxlen =21	Maxlen =26	Maxlen =31	Maxlen =36	Maxlen =41	Maxlen =46	Maxlen =51
■ GSgrow	51	52	52	53	53	53	53
■ NetMining-B	347	452	489	504	503	500	504
■ NetMining-D	347	462	495	503	498	502	501
■ NOSEP-Back	234	290	301	304	305	304	308
■ NOSEP	233	287	301	300	305	302	305

Fig. 13: Comparison of the mining speed under different lengths in DNA sequences (s)

meeting the constraint and mining time will increase with the increase in $maxlen$. On the contrary, 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 XIII that when $maxlen$ exceeds 13, no matter how much $maxlen$ increases, the results are the same for all mining algorithms. Meanwhile, according to Figure 14, we can see the running time is a little bit different when $maxlen$ exceeds 13. All the five algorithms have the same phenomena. The reason lies that the running time will be a little bit different even though running the same instances twice. However 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 seen in Table XII.

TABLE XIII: 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
NetMining-B	258	308	422	584	672	672	672
NetMining-D	258	308	422	584	672	672	672
NOSEP-Back	258	308	422	584	672	672	672
NOSEP	258	308	422	584	672	672	672



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

Fig. 14: Comparison of the mining speed under different lengths in protein databases (s)

VI. CONCLUSIONS

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: no condition, the one-off condition, and the non-overlapping 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 a non-overlapping sequence pattern mining task with gap constraints, where a non-overlapping sequence pattern allows sequence letters to be utilized flexibly for pattern discovery. An effective mining algorithm, NOSEP, is proposed to solve non-overlapping sequence pattern mining with gap constraints. 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.

Many possible future works could be investigated. 1) In this paper, we focus on mining frequent sequence patterns, but for a real task, it is difficult to set the support threshold $minsup$. If $minsup$ is too big, there will be no frequent pattern. Conversely, if $minsup$ is too small, there will be many frequent patterns. Therefore a possible future work will be mining Top-k frequent sequence patterns. 2) This paper is a kind of user-specified gap requirements research. However, sometimes users do not have the knowledge to set the gaps. How to deal with without involving user-specified gaps could be explored. 3) How to handle the incremental sequence pattern mining under the non-overlapping condition should also be considered. 4) 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. 5) In this paper, two kinds of experiments are employed to exhibit that the non-overlapping condition has better performance than no condition and the one-off condition. However, 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.

References

[1] Li C, Yang Q, Wang J, Li M. Efficient mining of gap constrained subsequences and its various applications. *ACM Transactions on Knowledge Discovery from Data*, 2012, 6(1):2.

[2] Le B, Tran MT, Vo B. Mining frequent closed inter-sequence patterns efficiently using dynamic bit vectors. *Applied Intelligence*, 2015, 43(1): 74-84.

[3] Zhang S, Du Z, Wang J T L. New techniques for mining frequent patterns in unordered trees. *IEEE Transactions on Cybernetics*, 2015, 45(6): 1113-1125.

[4] Zhang L, Luo P, Tang L, et al. Occupancy-based frequent pattern mining. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2015, 10(2): 14.

[5] Min F, Wu Y, Wu X. The apriori property of sequence pattern mining with wildcard gaps. *International Journal of Functional Informatics and Personalised Medicine*, 2012, 4(1): 15-31.

[6] Tan CD, Min F, Wang M, et al. Discovering patterns with weak-wildcard gaps. *IEEE Access*, 2016, 4: 4922-4932.

[7] Rasheed F, Alhajj R. A framework for periodic outlier pattern detection in time-series sequences. *IEEE Transactions on Cybernetics*, 2014, 44(5): 569-582.

[8] Jiang H, Zhang J, Ma H, et al. Mining authorship characteristics in bug repositories. *Science China Information Sciences*, 2017, 60(1): 012107:1-16.

[9] Fradkin D, Mrchen F. Mining sequential patterns for classification. *Knowledge and Information Systems*, 2015, 45(3): 731-749.

[10] Zhou C, Cule B, Goethals B. Pattern based sequence classification. *IEEE Transactions on Knowledge and Data Engineering*, 2016, 28(5): 1285-1298.

[11] Wu X, Zhu X, He Y, et al. PMBC: Pattern mining from biological sequences with wildcard constraints. *Computers in Biology and Medicine*, 2013, 43: 481-492.

[12] Liao V C C, Chen M S. DFSP: a Depth-First SPelling algorithm for sequential pattern mining of biological sequences. *Knowledge and Information Systems*, 2014, 38(3): 623-639.

[13] Yang H, Duan L, Hu B, et al. Mining top-k distinguishing sequential patterns with gap constraint. *Journal of Software*, 2015, 26(11): 2994-3009.

[14] Wu Y, Wang L, Ren J, et al. Mining sequential patterns with periodic wildcard gaps. *Applied Intelligence*, 2014, 41(1): 99-116.

[15] Wang HF, Duan L, Zuo J, et al. Efficient mining of distinguishing sequential patterns without a predefined gap constraint. *Chinese Journal of Computers*, 2016, 39(10): 1979-1991.

[16] Zhang M, Kao B, Cheung DW, Yip KY. Mining periodic patterns with gap requirement from sequences. *ACM Transactions on Knowledge Discovery from Data* , 2007, 1(2):7.

[17] Bille P, Grtz I L, Vildhj H W, et al. String matching with variable length gaps. *Theoretical Computer Science*, 2012, 443: 25-34

[18] Wu X, Qiang J P, Xie F. Pattern matching with flexible wildcards. *Journal of Computer Science and Technology*, 2014, 29(5): 740-750.

[19] Guo D, Hu X, Xie F, et al. Pattern matching with wildcards and gap-length constraints based on a centrality-degree graph. *Applied Intelligence*, 2013, 39(1): 57-74.

[20] Wu Y, Shen C, Jiang H, et al. Strict pattern matching under non-overlapping condition. *Science China Information Sciences*, 2017, 60 (1):012101:1-16.

[21] Ghosh S, Feng M, Nguyen H, et al. Risk prediction for acute hypotensive patients by using gap constrained sequential contrast patterns. *In AMIA Annual Symposium Proceedings American Medical Informatics Association*, 2014: 1748.

[22] Wang X, Duan L, Dong G, et al. Efficient mining of density-aware distinguishing sequential patterns with gap constraints. *In: International Conference Database Systems for Advanced Applications, Bali*, 2014: 372-387.

[23] Yen SJ, Lee YS. Mining non-redundant time-gap sequential patterns. *Applied Intelligence*, 2013, 39(4):727-738.

[24] Zhu X, Wu X. Mining complex patterns across sequences with gap requirements. *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007: 2934-2940.

[25] Lam H, Morchen F, Fradkin D, et al. Mining compressing sequential patterns. *Statistical Analysis and Data Mining*, 2013, 7: 34-52.

[26] Ding B, Lo D, Han J, et al. Efficient mining of closed repetitive gapped subsequences from a sequence database. *In: Proceedings of IEEE International Conference on Data Engineering, Shanghai*, 2009. 1024-1035.

[27] Feng Y, Ji M, Xiao J, et al. Mining spatial-temporal patterns and structural sparsity for human motion data denoising. *IEEE transactions on cybernetics*, 2015, 45(12): 2693-2706.

[28] Hui L, Chen Y C, Weng J T Y, et al. Incremental mining of temporal patterns in interval-based database. *Knowledge and Information Systems*, 2016, 46(2): 423-448.

[29] Nie L, Jiang H, Ren Z, et al. Query expansion based on crowd knowledge for code search. *IEEE Transactions on Services Computing*, 2016, 9(5): 771-783.

[30] Jiang H, Nie L, Sun Z, et al. ROSF: Leveraging information retrieval and supervised learning for recommending code snippets. *IEEE Transactions on Services Computing*, 2016. doi:10.1109/TSC.2016.2592909

[31] Wu Y, Wu X, Min F, et al. A Nettree for pattern matching with flexible wildcard constraints. *In: Proceeding of IEEE International Conference on Information Reuse and Integration, Las Vegas*, 2010. 109-114.

[32] Wu Y, Tang Z, Jiang H, et al. Approximate pattern matching with gap constraints. *Journal of Information Science*, 2016, 42(5): 639658.

[33] Wu Y, Fu S, Jiang H, et al. Strict approximate pattern matching with general gaps. *Applied Intelligence*, 2015, 42(3): 566-580.

[34] Warmuth MK, Haussler D. On the complexity of iterated shuffle. *Journal of Computer and System Sciences*, 1984, 28(3): 345-358.

[35] Chai X, Jia X, Wu Y, et al. Strict pattern matching with general gaps and one-off condition. *Journal of Software*, 2015, 26: 1096 -1112.

[36] Chen G, Wu X, Zhu X, et al. Efficient string matching with wildcards and length constraints. *Knowledge and Information Systems*, 2006, 10(4): 399-419.

[37] Wu Y, Wu X, Jiang H, et al. A heuristic algorithm for MPMGOOC. *Chinese Journal of Computers*, 2011, 34(8): 1452-1462.

[38] Chen Y, Keogh E, Hu B, et al. The UCR time series classification archive. *URL www.cs.ucr.edu/˜eamonn/time_series_data/.*

[39] Lin J, Keogh E, Lonardi S, et al. A symbolic representation of time series, with implications for streaming algorithms. *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, 2003: 2-11.

[40] Wittkop T, Baumbach J, Lobo F P, et al. Large scale clustering of protein sequences with FORCE-A layout based heuristic for weighted cluster editing. *BMC bioinformatics*, 2007, 8(1): 396.

[41] Fournier-Viger P, Gomariz A, Gueniche T, et al. SPMF: a Java open-source pattern mining library. *Journal of Machine Learning Research*, 2014, 15(1): 3389-3393.

[42] Lo D, Khoo S C, Liu C. Efficient mining of iterative patterns for software specification discovery. *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining* , 2007: 460-469.

[43] Bock C, Paulsen M, Tierling S, et al. CpG island methylation in human lymphocytes is highly correlated with DNA sequence, repeats, and predicted DNA structure. *PLoS Genetics*, 2006, 2(3) : 243-252.

Appendix A

Proof. ***Proof of Theorem 1:*** Our previous work [20] has shown that we can obtain a maximal non-overlapping set D that has k non-overlapping occurrences; that is, $\langle d_{1,1}, d_{1,2}, ...d_{1,m}\rangle$,$\langle d_{2,1}, d_{2,2}, ...d_{2,m}\rangle$,... $\langle d_{k,1}, d_{k,2}, ...d_{k,m}\rangle$, where $d_{h,j} < d_{h+1,j}$ and $1 \leq h < k$. That work has also shown that $\langle d_{k,1}, d_{k,2}, ...d_{k,m}\rangle$ can be replaced by the maximal occurrence $\langle f_{k,1}, f_{k,2}, ...f_{k,m}\rangle$. Now we will show that $\langle d_{1,1}, d_{1,2}, ...d_{1,m}\rangle$ can be replaced by the minimal occurrence $\langle g_{1,1}, g_{1,2}, ...g_{1,m}\rangle$.

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

Case 1. There exist j $(1 \leq j \leq m)$ that satisfy $d_{1,j} < g_{1,j}$. $\langle g_{1,1}, g_{1,2}, ...g_{1,m}\rangle$ is not a minimal occurrence. This contradicts the assumption that $\langle g_{1,1}, g_{1,2}, ...g_{1,m}\rangle$ 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, $d_{1,j} > g_{1,j}$. $\langle d_{1,1}, d_{1,2}, ...d_{1,m}\rangle$ and $\langle g_{1,1}, g_{1,2}, ...g_{1,m}\rangle$ are two non-overlapping occurrences. So there should be $k + 1$ non-overlapping occurrences. This contradicts the assumption that there are k non-overlapping 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 $\langle d_{1,1}, d_{1,2}, ...d_{1,m}\rangle$ and $\langle d_{i,1}, d_{i,2}, ...d_{i,m}\rangle$ $(1 < i \leq k)$ are two non-overlapping 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 $\langle g_{1,1}, g_{1,2}, ...g_{1,m}\rangle$ and $\langle d_{i,1}, d_{i,2}, ...d_{i,m}\rangle$ are two non-overlapping occurrences. Hence $\langle g_{1,1}, g_{1,2}, ...g_{1,m}\rangle$ can be used to replace $\langle d_{1,1}, d_{1,2}, ...d_{1,m}\rangle$.

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

Proof. ***Proof of Theorem 2:*** The prefix sub-pattern and the suffix sub-pattern 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 $sup(P, S) < minsup$. 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, the non-overlapping sequence pattern mining with gap constraints satisfies the Apriori property. □

Proof. ***Proof of Theorem 3:*** In NOSEP, the frequent patterns are stored in $meta$. Suppose there are F frequent patterns. So the space complexity of $meta$ is $O(m \times F)$. Since F is far less than n; that is $F << n$, the space complexity of $meta$ can be neglected. Now, we consider the space complexity of NETGAP. NETGAP 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. Hence, the space complexity of NETGAP and NOSEP are both $O(m \times n \times w)$ 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 complexity of NETGAP and NOSEP are both $O(m \times n \times w/r/r)$ in the average case. □

Proof. ***Proof of Theorem 4:*** It is easy to know that the time complexity of Algorithm 3 can be neglected. We consider the worst case at first. As we mentioned above, we can see that the time complexity of creating a Nettree is $O(m \times n \times w)$. There are no more than n non-overlapping occurrences. It is easy to see that no more than $O(m \times w)$ nodes in the m-th level can be pruned according Lemma 3, hence, there are no more than $O(m \times m \times w)$ can be deleted. Therefore, the time complexity of NETGAP is $O(m \times m \times N \times w)$ for an N-length sequence database. Since there are L candidate patterns, in the worst case, the time complexity of NOSEP is $O(m \times m \times N \times w \times L)$. Moreover, as mentioned in the space complexity, it is easy to see that in the average case, the time complexity of NOSEP is $O(m \times m \times N \times w \times L/r/r/r)$. □