SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

January 2017, Vol. 60 012101:1–012101:16
doi: 10.1007/s11432-015-0935-3

Strict pattern matching under non-overlapping condition

Youxi WU[1,2*], Cong SHEN[1,3], He JIANG[4] & Xindong WU[5,6]

[1]*School of Computer Science and Engineering, Hebei University of Technology, Tianjin* 300401*, China;*
[2]*Hebei Province Key Laboratory of Big Data Calculation, Tianjin* 300401*, China;*
[3]*School of Computer Science and Technology, Tianjin University, Tianjin* 300072*, China;*
[4]*School of Software, Dalian University of Technology, Dalian* 116621*, China;*
[5]*School of Computer Science and Information Engineering, Hefei University of Technology, Hefei* 230009*, China;*
[6]*School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA* 70503*, USA*

Received February 14, 2016; accepted May 11, 2016; published online November 15, 2016

Abstract Pattern matching (or string matching) is an essential task in computer science, especially in sequential pattern mining, since pattern matching methods can be used to calculate the support (or the number of occurrences) of a pattern and then to determine whether the pattern is frequent or not. A state-of-the-art sequential pattern mining with gap constraints (or flexible wildcards) uses the number of non-overlapping occurrences to denote the frequency of a pattern. Non-overlapping means that any two occurrences cannot use the same character of the sequence at the same position of the pattern. In this paper, we investigate strict pattern matching under the non-overlapping condition. We show that the problem is in P at first. Then we propose an algorithm, called NETLAP-Best, which uses Nettree structure. NETLAP-Best transforms the pattern matching problem into a Nettree and iterates to find the rightmost root-leaf path, to prune the useless nodes in the Nettree after removing the rightmost root-leaf path. We show that NETLAP-Best is a complete algorithm and analyse the time and space complexities of the algorithm. Extensive experimental results demonstrate the correctness and efficiency of NETLAP-Best.

Keywords pattern matching, sequential pattern mining, gap constraint, flexible wildcard, non-overlapping, occurrence, Nettree

Citation Wu Y X, Shen C, Jiang H, et al. Strict pattern matching under non-overlapping condition. Sci China Inf Sci, 2017, 60(1): 012101, doi: 10.1007/s11432-015-0935-3

1 Introduction

Nowadays how to find valuable information in data has become a hot topic. Many researches focus on solving the problem from diverse aspects including mining sequential patterns [1], academic social networks [2], xml query [3], bug triage [4], discovering activities [5], time series analysis [6], text mining [7], document retrieval [8], data preprocessing for classification [9], and network intrusion detection [10]. Pattern matching (or string matching [11]) has played an important role in solving the above issues.

* Corresponding author (email: wuc@scse.hebut.edu.cn)

	1	2	3	4	5
S	g	c	g	c	g
1st occurrence	g	c	g	.	.
2nd occurrence	g	c	.	.	g
3rd occurrence	g	.	.	c	g
4th occurrence	.	.	g	c	g

Figure 1 All occurrences of pattern P in sequence S.

Diverse pattern matching methods have been investigated such as string similarity search [12], windowed subsequence matching [13], circular string matching [14], approximate string matching [15], and string searching in large spatial databases [16]. Recently, a new kind of wildcard, named gap constraint (or flexible wildcard [17, 18]), is widely used in the tasks of pattern matching [19] and sequential pattern mining [20, 21], since it is not only more flexible than traditional wildcards such as "?" and "*", but also more easy to meet users' requirements. For instance, pattern matching with gap constraints now is an essential task in searching for RNA sequence-structure pattern [19]. Hence, many researchers have paid increasing attention to handling this kind of challenging issue. In this paper, we focus on pattern matching with gap constraints (or flexible wildcards) which can be used to calculate the support (or the number of occurrences) of a pattern. Calulating the support is one of the essential tasks for sequential patterns mining [22]. A pattern with gap constraints [23] can be written as $p_1[a_1, b_1]p_2[a_2, b_2]p_3 \cdots [a_{k-1}, b_{k-1}]p_k$, where a_i and b_i $(1 \leqslant i < k)$ are given integers and they are the minimal and maximal length of wildcards between p_i and p_{i+1}, respectively. Now, an illustrative example is given as follows.

Example 1. Suppose we have $P = p_1[a_1, b_1]p_2[a_2, b_2]p_3 =$ g[0,2]c[0,2]g and sequence $S = s_1s_2s_3s_4s_5 =$ gcgcg, all occurrences of P in S are shown in Figure 1.

A group of positions of the pattern in the sequence is used to represent an occurrence in this paper. Consequently, for example, the first occurrence is $\langle 1, 2, 3\rangle$, and others are $\langle 1, 2, 5\rangle$, $\langle 1, 4, 5\rangle$, and $\langle 3, 4, 5\rangle$.

When a_i is equal to b_i, it means that the number of wildcards "?" between p_i and p_{i+1} is a_i. When a_i and b_i are 0 and ∞, respectively, it means that there exists a wildcard "*" between p_i and p_{i+1}. Therefore, gap constraints can be used to express the traditional wildcards, while the traditional wildcards cannot be used to express gap constraints. Hence, a pattern with gap constraints is more flexible than those with "?" or "*". A pattern with gap constraints is applied not only in the area of pattern matching, such as music information retrieval [24, 25], computational biology [26], and text matching [27], but also in the area of sequential pattern mining [22, 28–32]. For instance, Ding et al. [22] focused on mining closed repetitive gapped sub-sequences under the non-overlapping condition, which is similar to the one-off condition[1)]. Under the non-overlapping condition, s_j cannot be reused by the same p_i. In Example 1, occurrences $\langle 1, 2, 3\rangle$, and $\langle 3, 4, 5\rangle$, do not hold under the one-off condition, since character "g" at position 3 of the sequence is used twice. However, $\langle 1, 2, 3\rangle$, and $\langle 3, 4, 5\rangle$, are two occurrences under the non-overlapping condition, since character "g" at position 3 matches p_3 and p_1 in $\langle 1, 2, 3\rangle$, and $\langle 3, 4, 5\rangle$, respectively. To achieve the goal of sequential pattern mining, Ding et al. [22] proposed an effective matching algorithm, named INSgrow, to calculate the number of occurrences of a pattern. The following example is used to illustrate the principle of INSgrow.

Example 2. Given the same pattern $P =$ g[0,2]c[0,2]g and sequence $S =$ gcgcg as in Example 1, we show how INSgrow calculates the occurrences under the non-overlapping condition.

Here, we claim that I is the maximum non-overlapping occurrences of a pattern and sup(P) is the size of I [22]. INSgrow firstly obtains set I^{g}, which is the set of occurrences of pattern "g". Since s_1 is g, $\langle 1\rangle$ is an occurrence of pattern g. Similarly, $\langle 3\rangle$ and $\langle 5\rangle$ are two other occurrences. Hence, sup(g) is 3 (shown

1) The one-off condition is also called the strong version of the non-overlapping condition which was proved to be NP-complete by Ding et al. [22].

Table 1 Pattern growth from g to g[0,2]c[0,2]g

Set I^{g}	Set $I^{\mathrm{g[0,2]c}}$	Set $I^{\mathrm{g[0,2]c[0,2]g}}$
$\langle 1\rangle$	$\langle 1,2\rangle$	$\langle 1,2,3\rangle$
$\langle 3\rangle$	$\langle 3,4\rangle$	$\langle 3,4,5\rangle$
$\langle 5\rangle$		
sup(g) = 3	sup(g[0,2]c) = 2	sup(g[0,2]c[0,2]g) = 2

Table 2 Pattern growth from g to g[0,1]c[0,1]g

Set I^{g}	Set $I^{\mathrm{g[0,1]c}}$	Set $I^{\mathrm{g[0,1]c[0,1]g}}$
$\langle 1\rangle$	$\langle 1,2\rangle$	
$\langle 5\rangle$		
sup(g) = 2	sup(g[0,1]c) = 1	sup(g[0,1]c[0,1]g) = 0

in the first column of Table 1). Based on set I^{g}, INSgrow calculates set $I^{\mathrm{g[0,2]c}}$. We know that $\langle 1,2\rangle$ is the first occurrence with gap constraint [0,2] and $\langle 3,4\rangle$ is the second occurrence. Therefore, sup(g[0, 2]c) is 2. Finally, INSgrow gets set $I^{\mathrm{g[0,2]c[0,2]g}}$ based on $I^{\mathrm{g[0,2]c}}$, which is $\{\langle 1,2,3\rangle, \langle 3,4,5\rangle\}$.

We can see that there are exponential candidate occurrences in Example 1. If a backtracking strategy is used to solve this challenging issue, the time complexity of the algorithm will be exponential since the algorithm may sometimes check the sub-occurrences one by one. INSgrow adopts the leftmost principle to avoid backtracking. Unfortunately, INSgrow may lose some feasible occurrences. We show this case in Example 3.

Example 3. Given sequence S = gccag and pattern P = g[0, 1]c[0, 1]g, INSgrow builds set I^{g} which is $\{\langle 1\rangle, \langle 5\rangle\}$. Since the first "c" after s_1 is 2, $\langle 1,2\rangle$, subject to gap constraint [0,1], is the first occurrence of pattern g[0,1]c. Since the length of S is 5, there is no occurrence of pattern g[0,1]c beginning with $\langle 5\rangle$. Therefore, INSgrow obtains set $I^{\mathrm{g[0,1]c}}$ which is $\{\langle 1,2\rangle\}$. Since there is no occurrence of pattern P which begins with $\langle 1,2\rangle$, INSgrow calculates sup(g[0, 1]c[0, 1]g) which is 0. The results are shown in Table 2. However, we can see that there is an occurrence $\langle 1,3,5\rangle$ of pattern P in sequence S. Hence, INSgrow may lose some feasible occurrences.

The contributions of this paper are as follows.

(1) We present the definition of strict pattern matching under the non-overlapping condition, though Ding et al. [22] first proposed the non-overlapping condition, which mainly aimed at dealing with sequential pattern mining.

(2) We show that strict pattern matching under the non-overlapping condition is in P.

(3) We propose an effective algorithm, named NETLAP-Best, which employs a Nettree. NETLAP-Best iteratively calculates a rightmost root-leaf path, which is an occurrence under the non-overlapping condition and effectively prunes the useless nodes after removing the rightmost root-leaf path. We prove that NETLAP-Best is a complete algorithm and also analyse the time and space complexities of NETLAP-Best.

(4) The experimental results show the importance of the pruning strategy through the comparison between NETLAP-Best and NETLAP-Nonpruning. The consistent results of NETLAP-Best and NETLAP-Slow, which prunes the useless nodes on the whole Nettree, demonstrate the correctness of NETLAP-Best. Extensive comparative results also validate that NETLAP-Best has a better performance than other competitive algorithms.

The paper is organized as follows. Section 2 introduces related work. Section 3 presents the problem definition and proves that the problem is in P. Section 4 proposes NETLAP-Best, proves that the algorithm is complete, and analyses the time and space complexities of the algorithm. An illustrative example is also given in this section. Section 5 shows the performance of NETLAP-Best. We conclude this paper in Section 6.

Table 3 Different types of related work

	Mining/Matching	Strict/Loose	Type of condition	Length constraints
El-Ramly et al. [33]	Sequential pattern mining	Loose	-[a)]	No
Bille et al. [34]	Pattern matching	Loose/Strict	-/No[b)]	No
Li et al. [1] Zhang et al. [28] Wang et al. [20]	Sequential pattern mining	Strict	No	No
Wu et al. [18,35,36]	Pattern matching	Strict	No	No
Wu et al. [30] Lam et al. [32]	Sequential pattern mining	Strict	One-off	No
Chai et al. [37] Guo et al. [38] Wu et al. [39]	Pattern matching	Strict	One-off	Yes
Ding et al. [22]	Sequential pattern mining	Strict	Non-overlapping	Yes
This paper	Pattern matching	Strict	Non-overlapping	Yes

a) "-" means that we do not consider this.
b) Bille et al. [34] proposed two algorithms to deal with loose and strict pattern matching, respectively.

2 Related work

Considering that pattern matching is related to sequential pattern mining, we therefore make a comparison of related sequential pattern mining in Table 3.

Pattern matching with gap constraints can be divided into two types, loose pattern matching and strict pattern matching [35]. In strict pattern matching we use a group of positions of the pattern in the sequence to represent an occurrence, while in loose pattern matching the end position (or the beginning and end positions) of the pattern in the sequence is represented as an occurrence. In Example 1, occurrence $\langle 1,2,3\rangle$ is just the expression in strict pattern matching. In loose pattern matching, there are only two occurrences 3 and 5, i.e., there will emerge occurrences at positions 3 and 5. Obviously, strict pattern matching is more challenging than loose pattern matching, since strict pattern matching considers the matching process while loose pattern matching ignores that.

There are three kinds of conditions in strict pattern matching: no condition [18, 35, 36], the non-overlapping condition [22], and the one-off condition [37–39] which is called the stronger version of the non-overlapping condition in [22]. Under no condition, there are 4 occurrences, $\langle 1,2,3\rangle$, $\langle 1,2,5\rangle$, $\langle 1,4,5\rangle$, and $\langle 3,4,5\rangle$ in Example 1. In Section 1, we have introduced that there are 2 occurrences under the non-overlapping condition, $\langle 1,2,3\rangle$ and $\langle 3,4,5\rangle$. But under the one-off condition, there is only one, which can be any one selected from the above four occurrences.

All these conditions were applied in sequential pattern mining tasks. For instance, Zhang et al. [28] addressed sequential pattern mining with periodic gaps under no condition. To solve this problem more effectively, Wu et al. [29] proposed a more effective algorithm, which employed a pattern matching approach. Both Wu et al. [30] and Lam et al. [32] explored frequent patterns under the one-off condition. Ding et al. [22] proposed a new sequential pattern mining under the non-overlapping condition.

The span of an occurrence is the distance between the beginning and end positions of the pattern in the sequence. If the span of an occurrence is between two given integers, which are called the length constraints, then we say that the occurrence satisfies the length constraints. The span of $\langle 1,2,3\rangle$ is $3-1+1=3$. The length constraints are always used in different applications. For instance, the length constraints can express a time interval in a transactional database sequence.

From Table 3, we can figure out that our work dealing with strict pattern matching under the non-overlapping condition is an essential task in sequential pattern mining, as was addressed by Ding et al. [22]. It is a challenging issue for the following reasons.

(1) Backtracking strategy cannot be used. INSgrow [22] without using backtracking strategy may lose some feasible occurrences according to Example 3. We know that $\langle 1,3,5\rangle$ is the solution for that instance.

When we find that there is no occurrence beginning with $\langle 1,2\rangle$, we have to return back to $\langle 1\rangle$ and get the next sub-occurrence $\langle 1,3\rangle$. Then we can obtain the occurrence $\langle 1,3,5\rangle$. So backtracking strategy is used in this method. Zhang et al. [28] proved that there are exponential occurrences with the same beginning position in the sequence. The time complexity of the algorithm using backtracking strategy is also exponential. So we need to propose a new algorithm without using backtracking strategy, which can also find $\langle 1,3,5\rangle$ effectively in Example 3.

(2) Obviously, the solution of strict pattern matching under no condition can be seen as a universal set, while the solutions of strict pattern matching under the non-overlapping condition and the one-off condition are two different subsets. The size of the universal set is exponential. So there will be more candidate solutions. Therefore, it is difficult for us to design an algorithm, which only uses one calculation to solve the problem. A feasible method is to find the non-overlapping occurrences one by one. When we deal with the one-off condition, after getting an occurrence, we can use an unmatchable character 'X' to replace the corresponding character in the sequence, but it fails to deal with the non-overlapping condition. The following example is used to illustrate the phenomenon.

Example 4. Given sequence S = agagaga and pattern P = a[0, 2]g[0, 2]a, we can see that there are two occurrences under the one-off condition, i.e., $\langle 1,2,3\rangle$ and $\langle 5,6,7\rangle$. After the first occurrence $\langle 1,2,3\rangle$ is found, S can be replaced by S' = XXXgaga. Then we can find the second occurrence $\langle 5,6,7\rangle$ in S'. However, there are three occurrences under the non-overlapping condition, i.e., $\langle 1,2,3\rangle$, $\langle 3,4,5\rangle$, and $\langle 5,6,7\rangle$. Since $\langle 3,4,5\rangle$ cannot be found in S', the method cannot be employed.

(3) It is difficult to distinguish which characters can be reused. In Example 4, when we obtain $\langle 1,2,3\rangle$, $\langle 1\rangle$ cannot be reused, but $\langle 3\rangle$ can be reused. So the new algorithm should distinguish the characters effectively.

3 Problem

3.1 Problem definition

Definition 1. A sequence S with length n is an ordered list of characters, which can be written as $S = s_1 s_2 \cdots s_i \cdots s_n$ $(1 \leqslant i \leqslant n)$.

Definition 2. A pattern with gap constraints can be written as $P = p_1[a_1, b_1]p_2 \cdots [a_j, b_j]p_{j+1} \cdots [a_{m-1}, b_{m-1}]\ p_m$ $(1 \leqslant j < m)$, where a_j and b_j are two given non-negative integers which refer to the lower bound and upper bound of the number of characters between p_j and p_{j+1}, respectively.

Apparently, if a_j or b_j is a negative integer, the gap is called a general gap [35,37], the problem is more general but also more difficult to tackle. In this paper, we only focus on non-negative gaps.

Definition 3. For all j $(1 < j \leqslant m)$, if there exists a group of positions $L = \langle l_1, l_2, \ldots, l_j, \ldots, l_m\rangle$, which satisfies character matching $(s_{l_j} = p_j)$, local gap constraint $(a_{j-1} \leqslant l_j - l_{j-1} - 1 \leqslant b_{j-1})$, and global length constraint (MinLen $\leqslant l_m - l_1 + 1 \leqslant$ MaxLen), then L is an occurrence of pattern P in sequence S. $l_m - l_1 + 1$ is span of L. $L[i, j] = \langle l_i, l_{i+1}, \ldots, l_j\rangle$ $(1 \leqslant i < j \leqslant m)$ is a sub-occurrence of sub-pattern $p_i[a_i, b_i]p_{i+1} \cdots [a_{j-1}, b_{j-1}]p_j$.

Definition 4. Let $L = \langle l_1, l_2, \ldots, l_j, \ldots, l_m\rangle$ and $L' = \langle l'_1, l'_2, \ldots, l'_j, \ldots, l'_m\rangle$ be two occurrences. If and only if $\forall 1 \leqslant j \leqslant m : l_j \neq l'_j$, L and L' are two non-overlapping occurrences.

Definition 5. A non-overlapping set is a subset of all the occurrences of pattern P in sequence S and any two occurrences in the set are non-overlapping. The problem of strict pattern matching under the non-overlapping condition refers to finding the maximum non-overlapping set C, i.e., a non-overlapping set with the largest number of occurrences.

Definition 6. Let $L = \langle l_1, l_2, \ldots, l_m\rangle$ be an occurrence. Compared with any other occurrence $L' = \langle l'_1, l'_2, \ldots, l'_m\rangle$, for all j $(1 \leqslant j \leqslant m)$, if it satisfies $l_j \geqslant l'_j$, then L is called maximal occurrence. Similarly, minimal occurrence can be defined.

Definition 7. Let k non-overlapping occurrences be $\langle d_{1,1}, d_{1,2}, \ldots, d_{1,m}\rangle$, $\langle d_{2,1}, d_{2,2}, \ldots, d_{2,m}\rangle, \ldots$, and

$\langle d_{k,1}, d_{k,2}, \ldots, d_{k,m}\rangle$. For all i and j, if $d_{i,j}$ is less than $d_{i+1,j}$, i.e., $d_{i,j} < d_{i+1,j}$, these occurrences are ordered non-overlapping occurrences.

Example 5. Given pattern $P = \text{g}[0,2]\text{c}[0,2]\text{g}$ and sequence $S = \text{gcgcg}$, all the occurrences of pattern P in sequence S are $\langle 1,2,3\rangle$, $\langle 1,2,5\rangle$, $\langle 1,4,5\rangle$, and $\langle 3,4,5\rangle$. Therefore, $\langle 1,2\rangle$ is a sub-occurrence. $\langle 1,2,3\rangle$ and $\langle 3,4,5\rangle$ are two ordered non-overlapping occurrences which are the minimal occurrence and maximal occurrence for the instance, respectively.

3.2 Problem analysis

Theorem 1. Let $\langle c_j, c_{j+1}\rangle$ and $\langle d_j, d_{j+1}\rangle$ be two sub-occurrences of sub-pattern $p_j[a_j, b_j]p_{j+1}$. Suppose $c_j < d_j$ and $c_{j+1} > d_{j+1}$, we can safely say that $\langle c_j, d_{j+1}\rangle$ and $\langle d_j, c_{j+1}\rangle$ are also two sub-occurrences. Conversely, if $c_j < d_j$ and $c_{j+1} < d_{j+1}$, then $\langle d_j, c_{j+1}\rangle$ may not be a sub-occurrence.

Proof. Since $\langle c_j, c_{j+1}\rangle$ and $\langle d_j, d_{j+1}\rangle$ are two sub-occurrences, it can be inferred that $p_j = s_{c_j} = s_{d_j}$, $p_{j+1} = s_{c_{j+1}} = s_{d_{j+1}}$, $a_j \leqslant c_{j+1} - c_j - 1 \leqslant b_j$ and $a_j \leqslant d_{j+1} - d_j - 1 \leqslant b_j$. There are two cases.

Case 1. Now we suppose $c_j < d_j$ and $c_{j+1} > d_{j+1}$. For $c_{j+1} > d_{j+1}$, it can be inferred that $a_j \leqslant d_{j+1} - d_j - 1 < c_{j+1} - d_j - 1$. For $c_j < d_j$, we have $c_{j+1} - d_j - 1 < c_{j+1} - c_j - 1 \leqslant b_j$. Therefore, $c_{j+1} - d_j - 1$ satisfies the local gap constraint, i.e., $a_j < c_{j+1} - d_j - 1 < b_j$. Hence $\langle d_j, c_{j+1}\rangle$ is a sub-occurrence. Similarly, we have $d_{j+1} - c_j - 1 < c_{j+1} - c_j - 1 \leqslant b_j$ and $a_j \leqslant d_{j+1} - d_j - 1 < d_{j+1} - c_j - 1$. Therefore, $d_{j+1} - c_j - 1$ satisfies the local gap constraint, i.e., $a_j < d_{j+1} - a_j - 1 < b_j$. Hence $\langle c_j, d_{j+1}\rangle$ is also a sub-occurrence.

Case 2. Now we suppose $c_j < d_j$ and $c_{j+1} < d_{j+1}$. We know that the lower bound of $d_{j+1} - d_j - 1$ is a_j. When $d_{j+1} - d_j - 1$ is equal to a_j, we have $c_{j+1} - d_j - 1 < d_{j+1} - d_j - 1 = a_j$. Therefore, $\langle d_j, c_{j+1}\rangle$ is not a sub-occurrence.

With case 1 and case 2, we complete our proof.

The following example is used to illustrate Theorem 1.

Example 6. Given sequence $S = \text{aaccgg}$ and pattern $P = \text{a}[0,2]\text{c}[0,1]\text{g}$, we know that both $\langle 1,4\rangle$ and $\langle 2,3\rangle$ are sub-occurrences of sub-pattern "a[0, 2]c". According to Theorem 1, we can safely say that both $\langle 1,3\rangle$ and $\langle 2,4\rangle$ are sub-occurrences. Similarly, we know that both $\langle 3,5\rangle$ and $\langle 4,6\rangle$ are sub-occurrences of sub-pattern "c[0, 1]g" while $\langle 3,6\rangle$ is not a sub-occurrence.

Theorem 2. Strict pattern matching under the non-overlapping condition is in P.

Proof. Suppose $\{\langle c_{1,1}, c_{1,2}, \ldots, c_{1,m}\rangle, \langle c_{2,1}, c_{2,2}, \ldots, c_{2,m}\rangle, \ldots, \langle c_{k,1}, c_{k,2}, \ldots, c_{k,m}\rangle\}$ is a maximum non-overlapping set for an instance, where for all $1 \leqslant i < k$ we have $c_{i,1} < c_{i+1,1}$ and k is no greater than n which means that there are no more than n non-overlapping occurrences since $1 \leqslant c_{i,1} < c_{i+1,1} \leqslant n$. All possible cases between $\langle c_{i,1}, c_{i,2}, \ldots, c_{i,m}\rangle$ and $\langle c_{i+1,1}, c_{i+1,2}, \ldots, c_{i+1,m}\rangle$ are shown as follows:

Case 1. $c_{i,j} < c_{i+1,j}$ and $c_{i,j+1} < c_{i+1,j+1}$. We do nothing in this case.

Case 2. $c_{i,j} < c_{i+1,j}$, $c_{i,j+1} > c_{i+1,j+1}$, and $c_{i,j+2} < c_{i+1,j+2}$ (where $1 \leqslant j \leqslant m-2$). According to Theorem 1, there are another two occurrences $\langle c_{i,1}, \ldots, c_{i,j}, c_{i+1,j+1}, c_{i,j+2}, \ldots, c_{i,m}\rangle$ and $\langle c_{i+1,1}, \ldots, c_{i+1,j}, c_{i,j+1}, c_{i+1,j+2}, \ldots, c_{i+1,m}\rangle$ (swapping $c_{i,j+1}$ and $c_{i+1,j+1}$), which replace the occurrences $\langle c_{i,1}, c_{i,2}, \ldots, c_{i,m}\rangle$ and $\langle c_{i+1,1}, c_{i+1,2}, \ldots, c_{i+1,m}\rangle$, respectively.

Case 3. $c_{i,j} < c_{i+1,j}$, $c_{i,j+1} > c_{i+1,j+1}$, $\ldots$, $c_{i,k-1} > c_{i+1,k-1}$, and $c_{i,k} < c_{i+1,k}$ (where $1 \leqslant j < k \leqslant m$). According to Theorem 1, there must exist two occurrences $\langle c_{i,1}, \ldots, c_{i,j}, c_{i+1,j+1}, \ldots, c_{i+1,k-1}, c_{i,k}, \ldots, c_{i,m}\rangle$ and $\langle c_{i+1,1}, \ldots, c_{i+1,j}, c_{i,j+1}, \ldots, c_{i,k-1}, c_{i+1,k}, \ldots, c_{i+1,m}\rangle$ (swapping $\langle c_{i,j+1}, \ldots, c_{i,k-1}\rangle$ and $\langle c_{i+1,j+1}, \ldots, c_{i+1,k-1}\rangle$), which replace the occurrences $\langle c_{i,1}, c_{i,2}, \ldots, c_{i,m}\rangle$ and $\langle c_{i+1,1}, c_{i+1,2}, \ldots, c_{i+1,m}\rangle$, respectively.

Case 4. $c_{i,j} < c_{i+1,j}$, $c_{i,j+1} > c_{i+1,j+1}$, $\ldots$, and $c_{i,m} > c_{i+1,m}$. According to Theorem 1, the instance has another two occurrences $\langle c_{i,1}, \ldots, c_{i,j}, c_{i+1,j+1}, \ldots, c_{i+1,m}\rangle$ and $\langle c_{i+1,1}, \ldots, c_{i+1,j}, c_{i,j+1}, \ldots, c_{i,m}\rangle$ (swapping $\langle c_{i,j+1}, \ldots, c_{i,m}\rangle$ and $\langle c_{i+1,j+1}, \ldots, c_{i+1,m}\rangle$), which replace the occurrences $\langle c_{i,1}, c_{i,2}, \ldots, c_{i,m}\rangle$ and $\langle c_{i+1,1}, c_{i+1,2}, \ldots, c_{i+1,m}\rangle$, respectively.

Now, we consider the global length constraint. The spans of the new occurrences in Case 1, Case 2, and Case 3 are not changed. Therefore, the new occurrences satisfy the global length constraint. In Case 4, the original occurrences satisfy the global length constraint, i.e., $\text{MinLen} \leqslant c_{i,m} - c_{i,1} + 1 \leqslant \text{MaxLen}$

and MinLen $\leqslant c_{i+1,m} - c_{i+1,1} + 1 \leqslant$ MaxLen. Since $c_{i,1} < c_{i+1,1}$ and $c_{i+1,m} > c_{i+1,m}$, so MinLen $< c_{i+1,m} - c_{i,1} + 1 <$ MaxLen and MinLen $< c_{i,m} - c_{i+1,1} + 1 <$ MaxLen, i.e., the new occurrences also satisfy the global length constraint.

We repeat the above process, until no pair of sub-occurrences can be exchanged. Since two occurrences $\langle c_{i,1}, c_{i,2}, \ldots, c_{i,m}\rangle$ and $\langle c_{i+1,1}, c_{i+1,2}, \ldots, c_{i+1,m}\rangle$ are replaced by two new occurrences according to the above cases. Therefore, the number of non-overlapping occurrences is neither increase nor decrease by this step. Now, we obtain a new maximum non-overlapping set D which also has k non-overlapping occurrences, i.e., $\langle d_{1,1}, d_{1,2}, \ldots, d_{1,m}\rangle$, $\langle d_{2,1}, d_{2,2}, \ldots, d_{2,m}\rangle, \ldots,$ and $\langle d_{k,1}, d_{k,2}, \ldots, d_{k,m}\rangle$. We can safely say that these occurrences are ordered non-overlapping occurrences.

Next, we will show that $\langle d_{k,1}, d_{k,2}, \ldots, d_{k,m}\rangle$ can be replaced by the maximal occurrence $\langle f_{k,1}, f_{k,2}, \ldots, f_{k,m}\rangle$.

(1) Suppose $\langle d_{k,1}, d_{k,2}, \ldots, d_{k,m}\rangle$ is the same as the maximal occurrence $\langle f_{k,1}, f_{k,2}, \ldots, f_{k,m}\rangle$. Of course, $\langle d_{k,1}, d_{k,2}, \ldots, d_{k,m}\rangle$ can be replaced by $\langle f_{k,1}, f_{k,2}, \ldots, f_{k,m}\rangle$ in this case.

(2) Assuming that $\langle d_{k,1}, d_{k,2}, \ldots, d_{k,m}\rangle$ and the maximal occurrence $\langle f_{k,1}, f_{k,2}, \ldots, f_{k,m}\rangle$ are different, three cases are given as follows.

Case 1. There exits j $(1 \leqslant j \leqslant m)$ satisfied $d_{k,j} > f_{k,j}$, which obviously contradicts the definition of maximal occurrence.

Case 2. For all j $(1 \leqslant j \leqslant m)$, $d_{k,j}$ is less than $f_{k,j}$, i.e., $d_{k,j} < f_{k,j}$. Apparently, $\langle d_{k,1}, d_{k,2}, \ldots, d_{k,m}\rangle$ and the maximal occurrence $\langle f_{k,1}, f_{k,2}, \ldots, f_{k,m}\rangle$ are two non-overlapping occurrences. So the problem has $k+1$ non-overlapping occurrences. But this contradicts the assumption that there are no more than k non-overlapping occurrences.

Case 3. For all j $(1 \leqslant j \leqslant m)$, $d_{k,j}$ is less than or equal to $f_{k,j}$, i.e., $d_{k,j} \leqslant f_{k,j}$. Therefore, we have $d_{i,j} < d_{k,j} \leqslant f_{k,j}$, where $i < k$. Hence, $\langle f_{k,1}, f_{k,2}, \ldots, f_{k,m}\rangle$ is different from other non-overlapping occurrences, i.e., $\langle d_{1,1}, d_{1,2}, \ldots, d_{1,m}\rangle$, $\langle d_{2,1}, d_{2,2}, \ldots, d_{2,m}\rangle, \ldots,$ and $\langle d_{k-1,1}, d_{k-1,2}, \ldots, d_{k-1,m}\rangle$. So $\langle d_{k,1}, d_{k,2}, \ldots, d_{k,m}\rangle$ can be replaced by the maximal occurrence $\langle f_{k,1}, f_{k,2}, \ldots, f_{k,m}\rangle$.

To sum up, no matter whether $\langle d_{k,1}, d_{k,2}, \ldots, d_{k,m}\rangle$ is the maximal occurrence or not, $\langle d_{k,1}, d_{k,2}, \ldots, d_{k,m}\rangle$ can be safely replaced by $\langle f_{k,1}, f_{k,2}, \ldots, f_{k,m}\rangle$. Therefore, the number of non-overlapping occurrences is neither increase nor decrease by this step according to the above three cases. Hence, new maximum non-overlapping set F also has k non-overlapping occurrences.

After removing $\langle d_{k,1}, d_{k,2}, \ldots, d_{k,m}\rangle$, we can also obtain other non-overlapping occurrences, since $\langle d_{k,1}, d_{k,2}, \ldots, d_{k,m}\rangle$ is different from other non-overlapping occurrences such as $\langle d_{1,1}, d_{1,2}, \ldots, d_{1,m}\rangle$, $\langle d_{2,1}, d_{2,2}, \ldots, d_{2,m}\rangle, \ldots,$ and $\langle d_{k-1,1}, d_{k-1,2}, \ldots, d_{k-1,m}\rangle$. Therefore, we can safely remove $\langle f_{k,1}, f_{k,2}, \ldots, f_{k,m}\rangle$. After removing $\langle f_{k,1}, f_{k,2}, \ldots, f_{k,m}\rangle$ and its overlapping occurrences, $\langle f_{k-1,1}, f_{k-1,2}, \ldots, f_{k-1,m}\rangle$ is the maximal occurrence. Iterating the above step, we can infer that $\langle d_{k-1,1}, d_{k-1,2}, \ldots, d_{k-1,m}\rangle$ can be replaced by $\langle f_{k-1,1}, f_{k-1,2}, \ldots, f_{k-1,m}\rangle$, which is different from other non-overlapping occurrences. Hence, we can conclude that for all i, $\langle d_{i,1}, d_{i,2}, \ldots, d_{i,m}\rangle$ can be replaced by $\langle f_{i,1}, f_{i,2}, \ldots, f_{i,m}\rangle$.

Consequently, to solve the problem of strict pattern matching under the non-overlapping condition, we select the maximal occurrence from all occurrences, and then delete it and its overlapping occurrences. We iterate this process until there is no occurrence. This strategy is called the maximal iterative selecting occurrence strategy. Obviously, we know that the minimal iterative selecting occurrence strategy can also solve the problem. This means that both of the strategies are feasible. All these steps mentioned above can be solved in P. Thus, Theorem 2 is proved.

The following example is used to show that any two of the non-overlapping occurrences can be changed into ordered non-overlapping occurrences.

Example 7. Given the same sequence $S =$ aaccgg and pattern $P =$ a$[0,2]$c$[0,1]$g as in Example 6, MinLen $= 4$, and MaxLen $= 6$, from Case 4 in Theorem 2 we know that $\langle 1,4,6\rangle$ and $\langle 2,3,5\rangle$ are two non-overlapping occurrences, so we can say that $\langle 1,3,5\rangle$ and $\langle 2,4,6\rangle$ are two ordered non-overlapping occurrences of the instance and the spans of two new occurrences are both 5 which satisfies the global length constraint.

In this paper, we design an effective algorithm, named NETLAP-Best, with the time complexity

$O(m \times m \times n \times W)$ in the worst case. However, experimental results show that it is faster than NETLAP-Nonpruning with the time complexity $O(m \times n \times W)$ in most cases. It will be introduced in the following section.

4 Proposed algorithm

4.1 Nettree

Definition 8. Nettree [18] is a kind of data structure that similar to tree, with such concepts as root, leaf, level, parent, child, etc. Nevertheless, evidently different from the tree structure, there are five characteristics which belong to the Nettree.

(1) A Nettree may have n roots, where $n \geqslant 1$.

(2) Any node except root may have more than one parent.

(3) The number of paths from one node to its ancestor may be more than one.

(4) The same node label can occur in different levels. n_j^i represents node i in the jth level.

(5) All parents of a node must be at the same level.

Definition 9. A path in a Nettree, from root to leaf, is called a root-leaf path.

Definition 10. Among the parents of node n_j^i, a parent with maximal label is called a rightmost parent. Likewise, a parent with minimal label is called a leftmost parent.

Definition 11. The maximal leaf in the mth level is called a max-leaf.

Definition 12. A root-leaf path which iterates the rightmost parent from the max-leaf to its root is called a rightmost root-leaf path. The length of a root-leaf path is m since the max-leaf is in the mth level.

Definition 13. A node can reach more than one root in a Nettree. The minimal root and maximal root of a node are called min-root and max-root, respectively. The min-root and max-root of a root are both themselves. The min-root and max-root of a node are the minimal value of its parents' min-roots and the maximal value of its parents' max-roots, respectively.

Lemma 1. Let A and B be two root-leaf paths without containing the same node. The corresponding occurrences of A and B are the non-overlapping occurrences.

Proof. A and B are $\langle n_1^{a_1}, n_2^{a_2}, \ldots, n_m^{a_m} \rangle$ and $\langle n_1^{b_1}, n_2^{b_2}, \ldots, n_m^{b_m} \rangle$, respectively. For all i $(1 \leqslant i \leqslant m)$, 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 non-overlapping occurrences.

Now, two examples are used to show that Nettree [18] is suitable to solve the problem. The first example is to show how to solve the problem without considering the global length constraint. Then we will show how to deal with the global length constraint effectively.

Example 8. Given sequence S = aggtaabgagaabb and pattern P = a[0,1]g[0,1]a[0,3]b, all occurrences which are $\langle 1, 3, 5, 7 \rangle$, $\langle 6, 8, 9, 13 \rangle$, $\langle 9, 10, 11, 13 \rangle$, $\langle 9, 10, 11, 14 \rangle$, and $\langle 9, 10, 12, 14 \rangle$ can be expressed by a Nettree which is shown in Figure 2(a).

Obviously, we can have a path from a leaf in the 4th level to a root, which is an occurrence without using backtracking strategy. For instance, when we get the leaf n_4^7, it is very easy for us to get the path $\langle n_1^1, n_2^3, n_3^5, n_4^7 \rangle$ or path $\langle 1, 3, 5, 7 \rangle$ in brief.

When we deal with the non-overlapping condition, each node in the Nettree can be used at most once. Therefore, it is easy to distinguish which characters can be reused. For instance, $\langle 6, 8, 9, 13 \rangle$ and $\langle 9, 10, 12, 14 \rangle$ are two non-overlapping occurrences because n_1^9 and n_3^9 are two different nodes. Meanwhile, $\langle 9, 10, 11, 13 \rangle$ and $\langle 9, 10, 12, 14 \rangle$ are overlapping occurrences, because they use the same nodes n_1^9 and n_2^{10}.

As analysed above, INSgrow [22] abandons the occurrence which begins with $\langle 1 \rangle$, since there is no occurrence that begins with sub-occurrence $\langle 1, 2 \rangle$. However, in Figure 2(a), it is easy to see that there is an occurrence $\langle 1, 3, 5, 7 \rangle$. So INSgrow may lose some feasible occurrences. INSgrow also begins with 5, 11, and 12 to probe an occurrence in Example 8. But in Nettree, it can be explicitly asserted that we

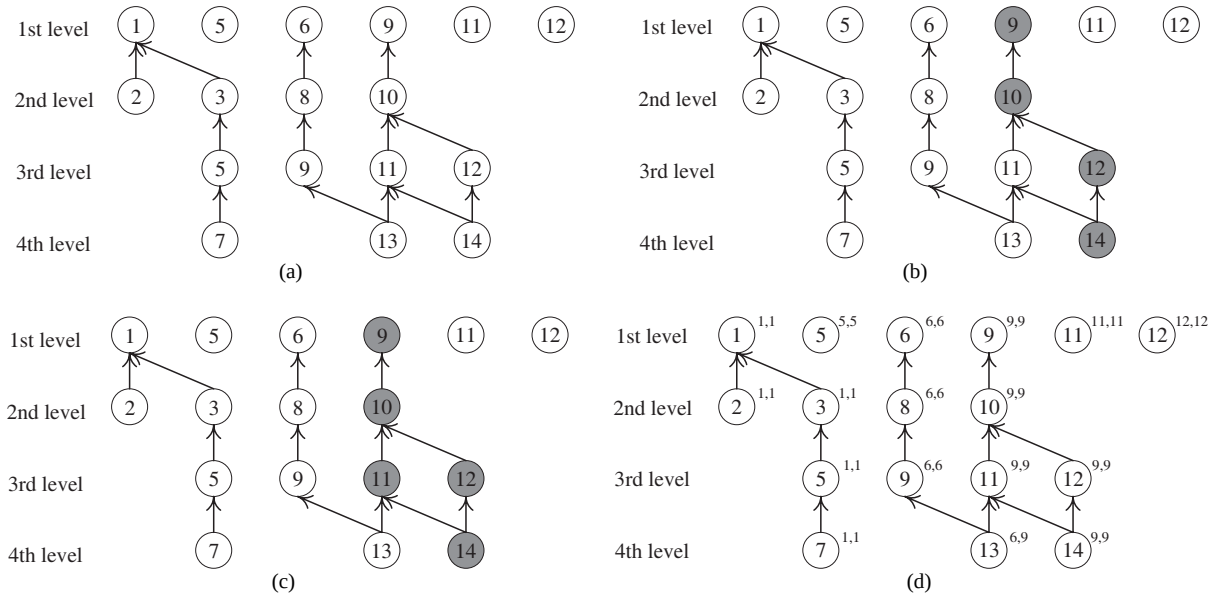

Figure 2 A Nettree for strict pattern matching with gap constraints. (a) A Nettree; (b) the new Nettree after pruning ⟨9,10,12,14⟩; (c) the new Nettree after pruning n_3^{11}; (d) a Nettree with min-root and max-root. Note: The grey nodes are pruned.

cannot probe from these nodes to find feasible occurrences. Therefore, the algorithm using Nettree can avoid not only missing occurrences such as $\langle 1,3,5,7\rangle$ in Example 8, but also invalid probing. Therefore, the algorithm using Nettree will be more effective than INSgrow.

Now, Example 8 is used to show how to solve the problem effectively. First, we know that the maximal leaf in the 4th level in Figure 2(a) is n_4^{14}. We can obtain path $\langle 9,10,12,14\rangle$ which iterates the parent with the maximal label from n_4^{14} to a root. Then we prune path $\langle 9,10,12,14\rangle$ and the new Nettree is shown in Figure 2(b).

Now, the maximal leaf in the 4th level in Figure 2(b) is n_4^{13}. Node n_3^{11} is the rightmost parent of n_4^{13}. However, n_3^{11} has no parent. Therefore, we also prune the nodes from the 2-nd level to the mth level which have no parent. For instance, n_3^{11} has no parent, we prune it. Then n_4^{13} has a parent n_3^9 and we cannot prune it. After pruning the node, the new Nettree is shown in Figure 2(c). We iterate this process and can obtain paths $\langle 6,8,9,13\rangle$ and $\langle 1,3,5,7\rangle$. Hence, there are three non-overlapping occurrences $\langle 1,3,5,7\rangle$, $\langle 6,8,9,13\rangle$, and $\langle 9,10,12,14\rangle$ in Example 8.

Please note that the pruning step is necessary. If n_3^{11} is not pruned, n_3^{11} is the rightmost parent of n_4^{13} and we cannot find an occurrence with nodes n_3^{11} and n_4^{13} in Figure 2(b). Then, we will face the similar problem of INSgrow. If we throw node n_4^{13} and will miss a valid occurrence $\langle 6,8,9,13\rangle$. Otherwise, we have to use backtracking strategy to find occurrence $\langle 6,8,9,13\rangle$. However, as we know that backtracking strategy cannot be to solve the problem and the reason has been explained in Section 1.

Example 9. In this example, the global length constraint is considered. The sequence and pattern are same as in Example 8, MinLen = 7 and MaxLen = 8. A Nettree with min-root and max-root is shown in Figure 2(d). The min-root and max-root are given at the right-top of each node.

We know that the min-root and max-root of node n_4^{14} are both 9. So $14-9+1=6$ is less than MinLen = 7. Therefore, there is no occurrence with leaf n_4^{14}. Then we deal with leaf n_4^{13}. The min-root and max-root of node n_4^{13} are 6 and 9, respectively. Since $13-6+1=8$ is greater than MinLen and no greater than MaxLen, there should be an occurrence which satisfies the global length constraints. Node n_4^{13} has two parents, node n_3^9 and node n_3^{11}. Although, node n_3^{11} is the rightmost parent of n_4^{13}, node n_4^{13} cannot select node n_3^{11} as its parent, because the min-root and max-root of node n_3^{11} are both 9 and $13-9+1=5$ is less than MinLen. Node n_4^{13} selects node n_3^9 as its parent. Then occurrence $\langle 6,8,9,13\rangle$ can be gotten. We can know that there are two occurrences in this example, $\langle 1,3,5,7\rangle$ and $\langle 6,8,9,13\rangle$. Therefore, it is easy to deal with the global length constraints using the min-root and max-root.

4.2 Solving algorithm

In this paper, we propose an algorithm, named NETLAP-Best, which can effectively handle the problem of strict pattern matching under the non-overlapping condition, is given in Algorithm 1.

Algorithm 1 NETLAP-Best

Require: Pattern P, sequence S and the length constraints (MinLen and MaxLen)
Ensure: Set C of the non-overlapping occurrences
1: Use Algorithm 2 to create a $Nettree$;
2: **for** l = the number of nodes in the mth level downto 1 step -1 **do**
3: Use Algorithm 3 to obtain the rightmost root-leaf path F;
4: $C = C \cup F.nodelabel$;
5: Update the $Nettree$ according to F using Algorithm 4;
6: **end for**
7: **return** C

Although our previous work also use Nettree to solve pattern matching problem, they are different from this paper. For instance, we use single-root Nettree to solve approximate strict pattern matching with non-negative gap constraints [36]. In order to solve general gap constraints, a Nettree can be created by scanning the sequence m times [35, 37]. In this paper, we focus on non-negative gap constraints, a Nettree can be created by only scanning the sequence once. The principle of creating a Nettree according to pattern P and sequence S is given as follows. If s_i is equal to p_j and there is a node which satisfies the local constraints with s_i in the $(j-1)$th level, then we will create node n_j^i in the jth level and find all parents of node n_j^i from the $(j-1)$th level. In order to avoid numerous invalid calculations, variable k in the algorithm is used to indicate the start position of the $(j-1)$th level. If a node is one of previous nodes of the kth node, then the node will not be a parent of n_j^i. At last, we calculate the min-root and the max-root of node n_j^i. An algorithm, named CreateNettree, is given in Algorithm 2.

Algorithm 2 CreateNettree

Require: Pattern P and sequence S
Ensure: $Nettree$
1: **for** $i = 1$ to n step 1 **do**
2: Let the start positions of each level T be 1;
3: **for** $j = 1$ to m step 1 **do**
4: **if** $p_j = s_i$ **then**
5: **if** $j = 1$ **then**
6: $Nettree[1]$.add(n_1^i);
7: The min-root and max-root of n_1^i are both i;
8: **else**
9: $k = T_{j-1}$; //k is used to indicate the start position of the $(j-1)$th level
10: **while** the gap between i and the kth node in the $(j-1)$th level $> a_{j-1}$ **do**
11: **if** the gap between i and the kth node in the $(j-1)$th level $> b_{j-1}$ **then**
12: T_{j-1} ++; //Update the start position of the $(j-1)$th level
13: continue;
14: **end if**
15: **if** n_j^i does not exist **then**
16: $Nettree[j]$.add(n_j^i);
17: **end if**
18: The kth node in the $(j-1)$th level is a new parent of n_j^i;
19: Update the min-root and max-root of n_j^i;
20: **end while**
21: **end if**
22: **end if**
23: **end for**
24: **end for**
25: **return** $Nettree$;

Algorithm 3, named Rightmost, can get the rightmost root-leaf path F.

Algorithm 3 Rightmost

Require: $Nettree$, m, l, and the global length constraint (MinLen and MaxLen)
Ensure: The rightmost root-leaf path F
1: **if** $Nettree[m][l].reused =$ true **then**
2: $F[m] = Nettree[m][l]$; //the max-leaf of Nettree in the mth level
3: $F[m].reused =$ false;
4: **for** $i = m$ downto 2 step -1 **do**
5: **for** $j = F[i].parentnumber$ downto 1 step -1 **do**
6: $node = F[i].parent[j]$;
7: **if** $node$ can be used and the spans between $F[m]$ and min-root or max-root of $node$ satisfy the global length constraint **then**
8: $F[i-1] = node$;
9: $F[i-1].reused =$ false;
10: break;
11: **end if**
12: **end for**
13: **end for**
14: **end if**
15: **return** F;

Lemma 2. If a node does not have its leftmost parent, it can be pruned.

Proof. Since we choose the rightmost root-leaf path to prune, the leftmost parent is the last parent which will be pruned. So if a node does not have its leftmost parent, after its rightmost root-leaf path is pruned, it will have no parent. Therefore, the node can be pruned.

In this paper, the property of reused is used to prune nodes simply. If reused of a node is false, which means that the node is pruned. Algorithm 4, named Pruning, employs Lemma 2 to prune other nodes which have no parent according to F.

Algorithm 4 Pruning

Require: $Nettree$, m, and F
Ensure: $Nettree$
1: **for** i=2 to $m-1$ step 1 **do**
2: **for** $position = F[i].position$ downto 1 step -1 **do**
3: $current = Nettree[i][position]$;
4: **if** $current.parent[1].reused =$ false **then**
5: $current.reused =$ false;
6: **else**
7: break; //find the first available node
8: **end if**
9: **end for**
10: **end for**
11: **return** $Nettree$;

In order to verify Algorithm 4 is necessary, we also propose an algorithm, named NETLAP-Nonpruning, which does not contain Algorithm 4 (Algorithm 1 does not contain line 5). We can safely say that NETLAP-Nonpruning will face the same problem as INSgrow. In order to show Algorithm 4 is effective, an algorithm with the time complexity $O(m \times W \times n^2)$, named NETLAP-Slow, which prunes the nodes having no parent on the whole Nettree is also proposed.

4.3 Algorithm analysis

Theorem 3. The rightmost root-leaf path is the maximal occurrence of the pattern in the sequence.

Proof. Obviously, $F[m]$ in Algorithm 3 is the max-leaf. Therefore, $F[m]$ is the maximal value of all occurrences. Now, we suppose $F[k]$ is the maximal value of all occurrences. We will prove that $F[k-1]$ is the maximal value of all occurrences, which is shown as follows.

Suppose $l[k-1]$ is the maximal value of all occurrences at position $k-1$ and $F[k-1]$ is less than $l[k-1]$, i.e., $F[k-1] < l[k-1]$. Let $l[k]$ be the maximal value of sub-occurrences with $l[k-1]$. There are three cases.

Case 1. $l[k] > F[k]$. But this contradicts the fact that $F[k]$ is the max value at position k in all occurrences.

Case 2. $l[k] < F[k]$. According to Theorem 1, $\langle l[k-1], F[k]\rangle$ is also a sub-occurrence in this case. This contradicts the assumption that $l[k]$ is the max value of sub-occurrences.

Case 3. $l[k] = F[k]$. So $\langle F[k-1], F[k]\rangle$ and $\langle l[k-1], F[k]\rangle$ are two sub-occurrences. But $F[k-1] < l[k-1]$ is contradicts the assumption that $F[k-1]$ is the rightmost parent of $F[k]$.

Therefore, $F[k-1]$ is the maximal value of all occurrences. Hence, the rightmost root-leaf path in Algorithm 3 is the maximal occurrence. Theorem 3 is proved.

Theorem 4. NETLAP-Best is complete.

Proof. According to Theorem 2, there are two strategies to solve the problem. NETLAP-Best employs Algorithm 3 to iteratively find the maximal occurrence. Therefore, NETLAP-Best is complete.

Theorem 5. The space complexity of NETLAP-Best is $O(m \times n \times W)$, where m, n, and W are the length of pattern and sequence, and maximal gap ($W = \max(b_i - a_i + 1)$), respectively.

Proof. A Nettree has m levels, each level has no more than n nodes, and each node has W children at most. Therefore, the space complexity of NETLAP-Best is $O(m \times n \times W)$ in the worst case. Theorem 5 is proved.

Theorem 6. The time complexity of NETLAP-Best is $O(m \times m \times n \times W)$.

Proof. We can see that the iteration times of line 1 and line 3 are $O(n)$ and $O(m)$, respectively. We can also know that the iteration times of line 10 is $O(W)$ since each node has no more than W parents. So the time complexity of Algorithm 2 is $O(m \times n \times W)$. We can also infer that the time complexity of Algorithm 2 is $O(m \times n \times W)$ from Theorem 5. The time complexity of Algorithm 3 is $O(m \times W)$ since each node has no more than W children. There are no more than $W \times (i-1)$ nodes in the ith level which have no parent. Therefore, Algorithm 4 checks no more than $W \times (m-1) \times m$ nodes. Hence, the time complexity of Algorithm 4 is $O(m \times m \times W)$. Since there are $n - m$ occurrences at most, therefore the time complexity of NETLAP-Best is $O(m \times n \times W + (n-m) \times (m \times W + m \times m \times W)) = O(m \times m \times n \times W)$ in the worst case. Hence, Theorem 6 is proved.

According to Theorem 6, we can know that the time complexity of NETLAP-Nonpruning is $O(m \times n \times W + (n-m) \times (m \times W)) = O(m \times n \times W)$ since NETLAP-Nonpruning does not employ Algorithm 4.

5 Performance evaluation

5.1 Experimental environment and data

All experiments are conducted on a computer with Intel(R) Core(TM)2 Duo CPU 1.99 GHz, 1.92 GB of RAM, and Windows XP SP3. To the best of our knowledge, there is no existing method tackling the same problem except INSgrow algorithm [22]. VC++ 6.0 is used to develop all algorithms, INSgrow, NETLAP-Nonpruning, NETLAP-Slow, and NETLAP-Best, which can be downloaded from http://wuc.scse.hebut.edu.cn/nettree/non-overlapping. To evaluate the performance and scalability of the algorithms, DNA and protein sequences are selected. In this paper, we use the patterns and sequences of DNA which were used in the problem of strict pattern matching under the one-off condition in literature [39]. These real DNA sequences can be downloaded from http://www.ncbi.nlm.nih.gov.

5.2 Experimental results

Figures 3 and 4 show the comparison of the results and the running time, respectively. We can analyse the results from the following aspects:

(1) NETLAP-Slow and NETLAP-Best are complete, while NETLAP-Nonpruning and INSgrow may lose some feasible occurrences. In Figure 3, NETLAP-Slow and NETLAP-Best obtain the same results on all the 72 instances. All the results are superior to those of NETLAP-Nonpruning and INSgrow. For instance, both NETLAP-Slow and NETLAP-Best obtain 125 occurrences of pattern $P2$ in $S1$, while

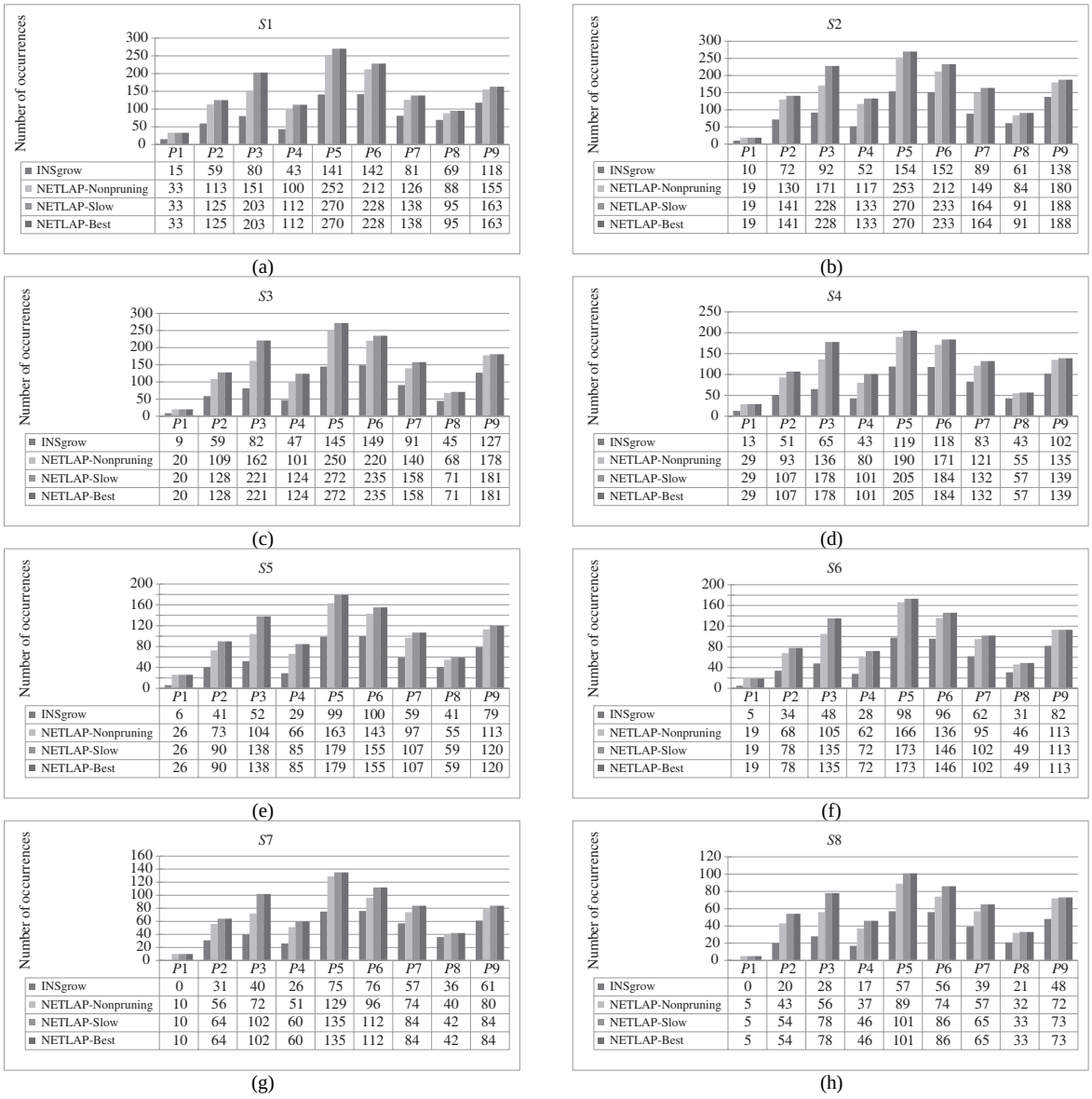

Figure 3 Comparison of results on DNA of (a) $S1$, (b) $S2$, (c) $S3$, (d) $S4$, (e) $S5$, (f) $S6$, (g) $S7$, (h) $S8$.

INSgrow and NETLAP-Nonpruning get 59 occurrences and 113 occurrences, respectively. Hence the extensive experimental results on biological data validate that both NETLAP-Nonpruning and INSgrow may lose some feasible occurrences, and NETLAP-Slow and NETLAP-Best are correct.

(2) Although both NETLAP-Nonpruning and INSgrow may lose some feasible occurrences, NETLAP-Nonpruning is superior to INSgrow. As mentioned above, we know that NETLAP-Nonpruning will face the same trouble as INSgrow. However, NETLAP-Nonpruning is better than INSgrow on all the 72 instances in Figure 3. The reason is that INSgrow seeks the minimal occurrence from up to down. From Figure 2, we can see that there are a lot of roots that cannot reach the leaves. Consequently, a large amount of the non-overlapping occurrences will be misjudged. NETLAP-Nonpruning does not prune nodes. So some nodes cannot reach a root. However, this phenomenon is far less than that of INSgrow. Hence, NETLAP-Nonpruning is superior to INSgrow.

(3) NETLAP-Best runs faster than NETLAP-Slow. From Figure 4, we know that NETLAP-Best runs faster than NETLAP-Slow on all the 72 instances. For example, the running time of NETLAP-Slow is 174 ms for pattern $P2$ in sequence $S3$, while NETLAP-Best is 45 ms. The reason for this situation is that NETLAP-Best employs a more effective pruning strategy than NETLAP-Slow.

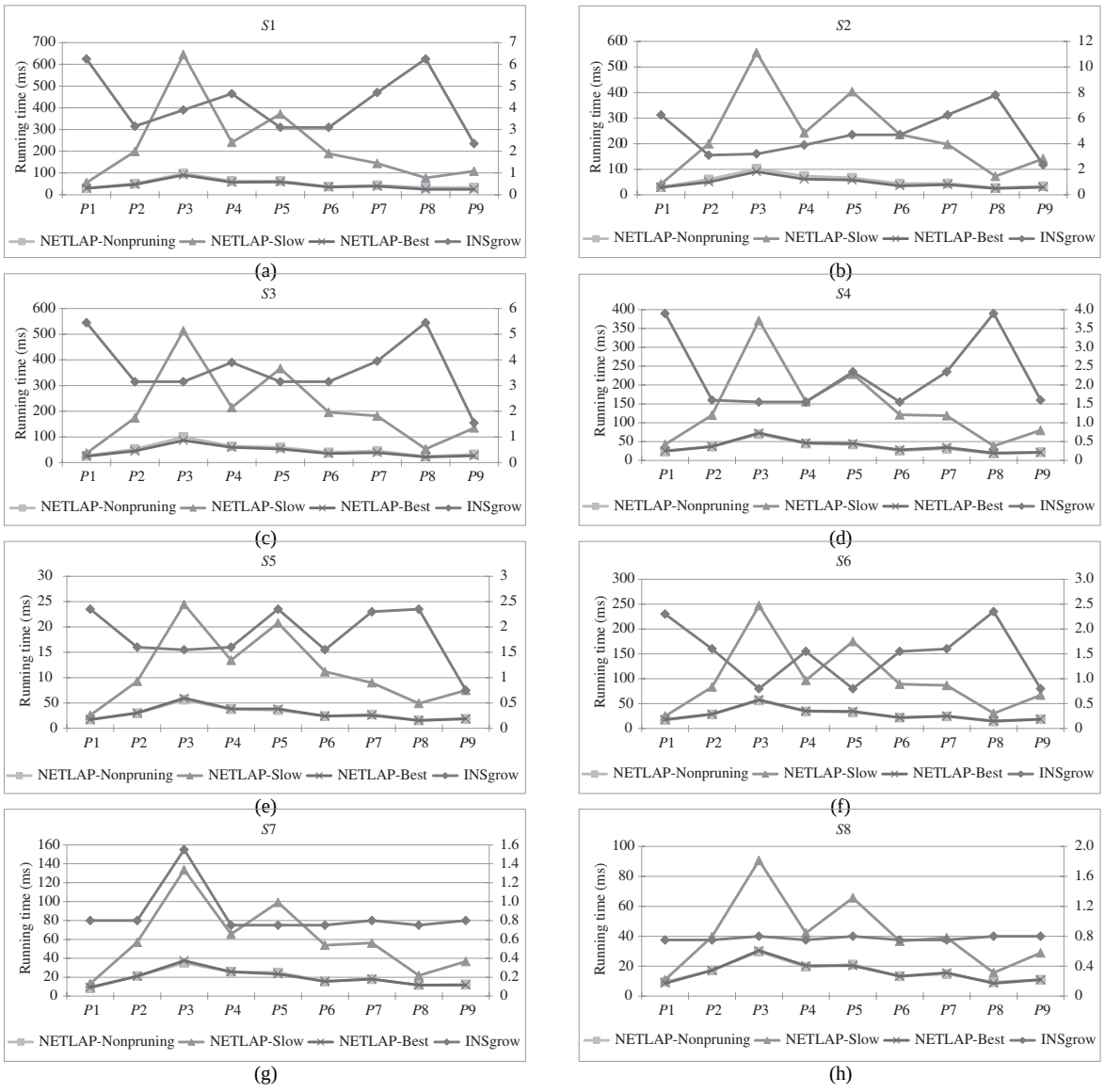

Figure 4 Comparison of running time on DNA of (a) $S1$, (b) $S2$, (c) $S3$, (d) $S4$, (e) $S5$, (f) $S6$, (g) $S7$, (h) $S8$.

(4) NETLAP-Best runs a bit faster than NETLAP-Nonpruning. The time complexity of NETLAP-Best is $O(m \times m \times n \times W)$ in the worst case, while the time complexity of NETLAP-Nonpruning is $O(m \times n \times W)$ as NETLAP-Nonpruning does not prune any node. It seems that NETLAP-Nonpruning should be much faster than NETLAP-Best. Experimental results show that NETLAP-Nonpruning is a little slower. Many instances in Figure 4 show that NETLAP-Best runs a little faster than NETLAP-Nonpruning. Nevertheless in some instances, NETLAP-Best is a bit slower. For example in $P2$–$S2$, NETLAP-Best takes 50 ms, while NETLAP-Nonpruning takes 60 ms. But in $P4$–$S6$, NETLAP-Best takes 35 ms, while NETLAP-Nonpruning only takes 34 ms. We calculate the total time of NETLAP-Best and NETLAP-Nonpruning on 72 cases in Figure 4 which are 2388 ms and 2513 ms, respectively. So NETLAP-Best is a little faster than NETLAP-Nonpruning actually. The reasons are that there are only a few nodes need to be pruned and NETLAP-Best prunes nodes effectively. After pruning, NETLAP-Best only seeks the maximal occurrences from valid leaves, while NETLAP-Nonpruning has to find the occurrences from all the leaves. However, some of them are invalid. For example, a Nettree has the first three levels in Figure 2(a), we can see that NETLAP-Nonpruning needs to search four times for the maximal occurrence, while NETLAP-Best just needs to search three times. Consequently, NETLAP-Best is faster than NETLAP-Nonpruning in most cases.

(5) INSgrow runs obviously faster than other three algorithms. In Figure 4, the running time of INSgrow refers to the time data on the right and the other algorithms refer to the left. The total time of INSgrow on 72 cases is 177.4 ms. So INSgrow is about 13.5 times faster than NETLAP-Best on DNA. The reason is that INSgrow is easy and may lost many feasible occurrences. While the other three algorithms are obviously more complex than INSgrow, in the case of less incorrectly probing, their time consumptions must be greater than INSgrow.

In summary, extensive comparative experimental results validate that NETLAP-Best, which is superior to the state-of-the-art algorithm, is more effective than other competitive algorithms.

6 Conclusion

In this paper, we propose strict pattern matching under the non-overlapping condition, which means that no character in the sequence can be reused by the same p_i. We prove that the problem is in P. A Nettree structure which may have more than one parent and more than one root is employed to solve the problem. Since the same node label can occur in different levels in a Nettree, this property can be utilized to deal with the non-overlapping condition effectively. Therefore, we propose NETLAP-Best to find the rightmost root-leaf path as a non-overlapping occurrence and effectively prune the nodes which cannot reach the root in a small range, and iterate this process. In this paper, we prove the completeness of NETLAP-Best, and also show that the time and space complexities are $O(m \times n \times W)$ and $O(m \times m \times n \times W)$, respectively, where m, n, and W are the length of pattern P, the length of sequence S, and the maximal gap of pattern P, respectively. We also construct NETLAP-Slow which prunes on the whole Nettree and NETLAP-Nonpruning which does not prune. Extensive comparative results on the real biological data show that NETLAP-Best has a better performance than other competitive algorithms.

The next step in this study is to apply the approach in this paper for the investigation of sequential pattern mining under the non-overlapping condition.

Acknowledgements The work was supported by National Natural Science Foundation of China (Grant Nos. 61229301, 61571180, 61370144), Natural Science Foundation of Hebei Province (Grant Nos. F2013202138, G2014202031), Graduate Student Innovation Program of Hebei Province (Grant No. 220056), and Youth Foundation of Education Commission of Hebei Province (Grant No. QN2014192).

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Li C, Yang Q Y, Wang J Y, et al. Efficient mining of gap-constrained subsequences and its various applications. ACM Trans Knowl Discov Data, 2012, 6: 2
2 Wang P, Xu B W, Wu Y R, et al. Link prediction in social networks: the state-of-the-art. Sci China Inf Sci, 2015, 58: 011101
3 Liu J, Ma Z M, Feng X. Answering ordered tree pattern queries over fuzzy XML data. Knowl Inf Syst, 2015, 43: 473–495
4 Xuan J F, Jiang H, Hu Y, et al. Towards effective bug triage with software data reduction techniques. IEEE Trans Knowl Data Eng, 2015, 27: 264–280
5 Cook D, Krishnan N C, Rashidi P. Activity discovery and activity recognition: a new partnership. IEEE Trans Cybern, 2013, 43: 820–828
6 Weng L N, Zhang P, Feng Z Y, et al. Short-term link quality prediction using nonparametric time series analysis. Sci China Inf Sci, 2015, 58: 082308
7 Rajpathak D, De S. A data-and ontology-driven text mining-based construction of reliability model to analyze and predict component failures. Knowl Inf Syst, 2016, 46: 87–113
8 Navarro G. Spaces, trees, and colors: the algorithmic landscape of document retrieval on sequences. ACM Comput Surv, 2014, 46: 52
9 Jiang H, Xuan J F, Ren Z L, et al. Misleading classification. Sci China Inf Sci, 2014, 57: 052106
10 Le H, Prasanna V K. A memory-efficient and modular approach for large-scale string pattern matching. IEEE Trans Comput, 2013, 62: 844–857
11 Claude F, Navarro G, Peltola H, et al. String matching with alphabet sampling. J Discrete Algorithms, 2012, 11: 37–50

12 Wandelt S, Deng D, Gerdjikov S, et al. State-of-the-art in string similarity search and join. ACM SIGMOD Rec, 2014, 43: 64–76
13 Li Z, Ge T J. Online windowed subsequence matching over probabilistic sequences. In: Proceedings of ACM International Conference on Management of Data. New York: ACM, 2012. 277–288
14 Chen K-H, Huang G-S, Lee R C-T. Bit-parallel algorithms for exact circular string matching. Comput J, 2014, 57: 731–743
15 Hu H, Wang H Z, Li J Z, et al. An efficient pruning strategy for approximate string matching over suffix tree. Knowl Inf Syst, 2016, 49: 121–141
16 Li F F, Yao B, Tang M W, et al. Spatial approximate string search. IEEE Trans Knowl Data Eng, 2013, 25: 1394–1409
17 Wu X D, Qiang J P, Xie F. Pattern matching with flexible wildcards. J Comput Sci Technol, 2014, 29: 740–750
18 Wu Y X, Wu X D, 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
19 Retwitzer M D, Polishchuk M, Churkin E, et al. RNAPattMatch: a web server for RNA sequence/structure motif detection based on pattern matching with flexible gaps. Nucleic Acids Res, 2015, doi: 10.1093/nar/gkv435
20 Wang X M, Duan L, Dong G Z, et al. Efficient mining of density-aware distinguishing sequential patterns with gap constraints. In: Proceedings of International Conference Database Systems for Advanced Applications, Bali, 2014. 372–387
21 Liao V C-C, Chen M-S. Efficient mining gapped sequential patterns for motifs in biological sequences. BMC Syst Biol, 2013, 7: S7
22 Ding B L, Lo D, Han J W, 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
23 Yang H, Duan L, Hu B, et al. Mining top-k distinguishing sequential patterns with gap constraint. J Softw, 2015, 26: 2994–3009
24 Crochemore M, Iliopoulos C, Makris C, et al. Approximate string matching with gaps. Nordic J Comput, 2002, 9: 54–65
25 Cantone D, Cristofaro S, Faro S. New efficient bit-parallel algorithms for the (δ, α)-matching problem with applications in music information retrieval. Int J Found Comput Sci, 2009, 20: 1087–1108
26 Cole J, Chai B, Farris R, et al. The Ribosomal Database Project (RDP-II): sequences and tools for high-throughput rRNA analysis. Nucleic Acids Res, 2005, 33: 294–296
27 Cole R, Gottlieb L, Lewenstein M. Dictionary matching and indexing with errors and don't care. In: Proceeding of Symposium on Theory of Computing, Chicago, 2004. 91–100
28 Zhang M H, Kao B, Cheung D W, et al. Mining periodic patterns with gap requirement from sequences. ACM Trans Knowl Discov Data, 2007, 1: 7
29 Wu Y X, Wang L L, Ren J D, et al. Mining sequential patterns with periodic wildcard gaps. Appl Intell, 2014, 41: 99–116
30 Wu X D, Zhu X Q, He Y, et al. PMBC: pattern mining from biological sequences with wildcard constraints. Comput Biol Med, 2013, 43: 481–492
31 Ibrahim A, Sastry S, Sastry P S. Discovering compressing serial episodes from event sequences. Knowl Inf Syst, 2016, 47: 405–432
32 Lam H, Mörchen F, Fradkin D, et al. Mining compressing sequential patterns. Stat Anal Data Min, 2013, 7: 34–52
33 El-Ramly M, Stroulia E, Sorenson P. From run-time behavior to usage scenarios: an interaction-pattern mining approach. In: Proceeding of ACM International Conference on Knowledge Discovery and Data Mining, Edmonton, 2002. 315–324
34 Bille P, Gørtz I, Vildhøj H W, et al. String matching with variable length gaps. Theor Comput Sci, 2012, 443: 25–34
35 Wu Y X, Fu S, Jiang H, et al. Strict approximate pattern matching with general gaps. Appl Intell, 2015, 42: 566–580
36 Wu Y X, Tang Z Q, Jiang H, et al. Approximate pattern matching with gap constraints. J Inf Sci, 2016, 42: 639–658
37 Chai X, Jia X F, Wu Y X, et al. Strict pattern matching with general gaps and one-off condition (in Chinese). J Softw, 2015, 26: 1096–1112
38 Guo D, Hu X G, Xie F, et al. Pattern matching with wildcards and gap-Length constraints based on a centrality-degree graph. Appl Intell, 2013, 39: 57–74
39 Wu Y X, Wu X D, Jiang H, et al. A heuristic algorithm for MPMGOOC. Chin J Comput, 2011, 34: 1452–1462