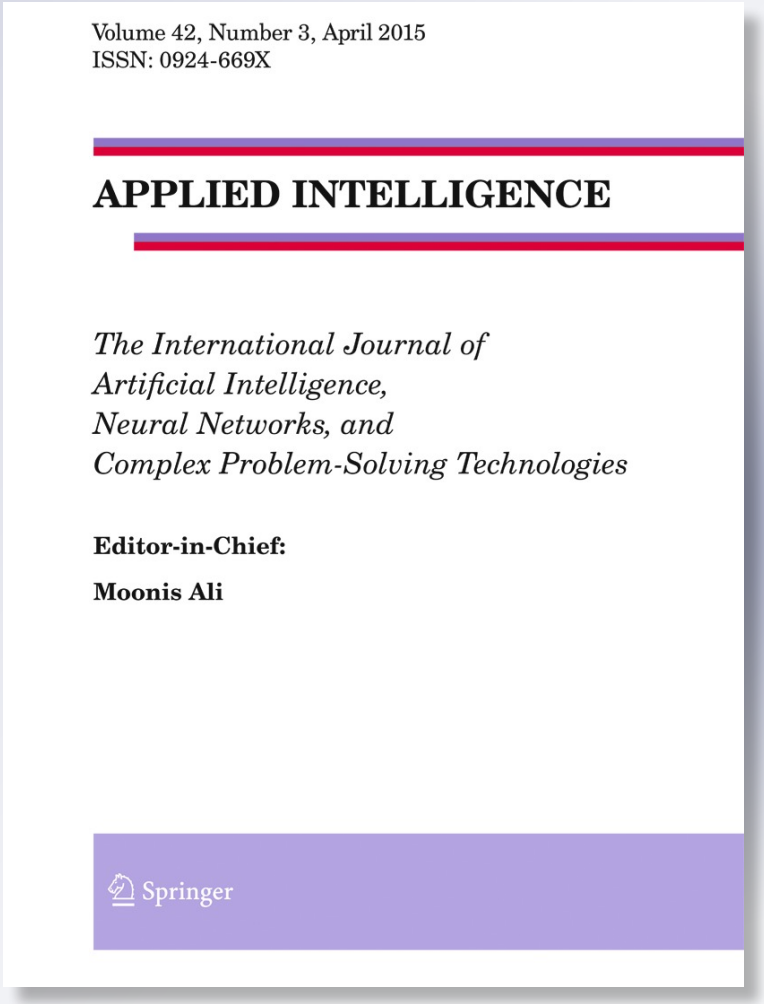
# *Strict approximate pattern matching with general gaps*

**Youxi Wu, Shuai Fu, He Jiang & Xindong Wu**

**Applied Intelligence**
The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies

ISSN 0924-669X
Volume 42
Number 3

Appl Intell (2015) 42:566-580
DOI 10.1007/s10489-014-0612-3




Springer

Appl Intell (2015) 42:566–580
DOI 10.1007/s10489-014-0612-3

# Strict approximate pattern matching with general gaps

**Youxi Wu · Shuai Fu · He Jiang · Xindong Wu**

Published online: 25 November 2014


**Abstract** Pattern matching with gap constraints is one of the essential problems in computer science such as music information retrieval and sequential pattern mining. One of the cases is called loose matching, which only considers the matching position of the last pattern substring in the sequence. One more challenging problem is considering the matching positions of each character in the sequence, called strict pattern matching which is one of the essential tasks of sequential pattern mining with gap constraints. Some strict pattern matching algorithms were designed to handle pattern mining tasks, since strict pattern matching can be used to compute the frequency of some patterns occurring in the given sequence and then the frequent patterns can be derived. In this article, we address a more general strict approximate pattern matching with Hamming distance, named SAP (Strict Approximate Pattern matching with general gaps and length constraints), which means that the gap constraints can be negative. We show that a SAP instance can be transformed into an exponential amount of the exact pattern matching with general gaps instances. Hence, we propose an effective online algorithm, named SETA (SubnETtree for sAp), based on the subnettree structure (a Nettree is an extension of a tree with multi-parents and multi-roots) and show the completeness of the algorithm. The space and time complexities of the algorithm are $O(m{\times}Maxlen{\times}W{\times}d)$ and $O(Maxlen{\times}W{\times}m^2{\times}n{\times}d)$, respectively, where $m$, $Maxlen$, $W$, and $d$ are the length of pattern $P$, the maximal length constraint, the maximal gap length of pattern $P$ and the approximate threshold. Extensive experimental results validate the correctness and effectiveness of SETA.

**Keywords** Approximate pattern matching · Hamming distance · General gap · Online algorithm

Y. Wu (✉) · S. Fu
School of Computer Science and Engineering, Hebei University of Technology, Tianjin, 300130, China
e-mail: wuc567@163.com

S. Fu
e-mail: 15822014723@126.com

X. Wu
e-mail: xwu@cems.uvm.edu

H. Jiang
School of Software, Dalian University of Technology, Dalian, 116621, China
e-mail: jianghe@dlut.edu.cn

X. Wu
School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, 230009, China

X. Wu
Department of Computer Science, University of Vermont, Burlington, VT, 05405, USA

## 1 Introduction

Pattern matching (also called string matching) is one of the essential problems in computer science with broad applications [1, 2]. The most classical pattern matching algorithm is KMP which was proposed by Knuth [3]. After that, Fischer and Paterson [4] first proposed pattern matching with wildcards (or 'don't care' symbols) and in their study the number of wildcards between two consecutive letters is a constant. Manber et al. [5] studied pattern matching with the gap constraint, which means that the wildcard is a range, but the pattern has only one gap constraint. In recent years, researchers have paid more attention to pattern matching with multiple gap

constraints, the pattern in this kind of issue can be described as $P = p_0[a_0, b_0]p_1...[a_j, b_j]p_{j+1}...[a_{m-2}, b_{m-2}]p_{m-1}$, where $a_j$ and $b_j$ are the minimal and maximal numbers that a wildcard can match between $p_j$ and $p_{j+1}$. Pattern matching with gaps has been applied to many domains. For instance, Navarro and Raffinot [6] proposed two algorithms which can be used for protein searching. Cole et al. [7] used the approximate pattern matching approach to judge whether the pattern string is in the specified text or dictionary. Crochemore et al. [8] investigated the $(C_m^T, \alpha)$-approximate matching which can be used in the music retrieval field. Cantone et al. [9] focused on the parallel-by-bit approach that can be applied in music information retrieval and analysis. In sequential pattern mining, Ji et al. [10] proposed the ConSGapMine algorithm which can mine minimal distinguishing sequences. Ferreira and Azevedo [11] proposed the gIL algorithm to mine protein sequences. Zhang et al. [12] proposed the MPP algorithm to mine sequential patterns with periodic gap constraints. Zhu and Wu [13] and Wu et al. [14] proposed state-of-the-art algorithms which have a better performance than that of MPP. All these researches mentioned above employed a pattern matching strategy. An illustrative example is given as follows to show all occurrences of a pattern with gaps in a sequence.

*Example 1* Given pattern $P = p_0[a_0, b_0]p_1[a_1, b_1]p_2 =$ C[0, 2]A[0, 2]G and sequence $S = s_0s_1s_2s_3s_4s_5 =$ CACAGG.

We know that CA..G is an occurrence which satisfies pattern $P$, and there are 5 occurrences like this. To denote conveniently, we use the subscripts of each character to represent an occurrence, hence CA..G can be denoted by $< 0, 1, 4 >$. Thus, the 5 occurrences of this problem can be described by $< 0, 1, 4 >$, $< 0, 3, 4 >$, $< 0, 3, 5 >$, $< 2, 3, 4 >$, and $< 2, 3, 5 >$. Nevertheless, $< 0, 1, 5 >$ is not an occurrence, because the gap between 1 and 5 is 3, which fails to satisfy the gap constraint [0,2]. We show all occurrences in Fig. 1. So in the sequential pattern mining task, we say the support of $P$ in $S$ is 5, while in the pattern matching issue, we say the number of occurrences of $P$ in $S$ is 5. Therefore, pattern matching plays an important role in sequential pattern mining with gaps, since one of the essential tasks of sequential pattern mining is to calculate the support of a pattern.

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Sequence | C | A | C | A | G | G |
| 1st occurrence | C | A | . | . | G | |
| 2nd occurrence | C | . | . | A | G | |
| 3rd occurrence | C | . | . | A | . | G |
| 4th occurrence | . | . | C | A | G | |
| 5th occurrence | . | . | C | A | . | G |

**Fig. 1** Strict pattern matching with gaps

Sequential pattern mining has very important applications in real problems. For instance, miners can discover the common sequential purchasing behaviours for most of the customers according to the transactional database [15]. A pattern can be that most of the customers purchased item A, after a while bought item B, and finally purchased item C. However, this kind of mining is under the non-negative gaps, which bounds the purchasing order of the consumers. Pattern ABC with non-negative gaps fails to be detected in the sequence containing BAC, but pattern ABC with general gaps occurs in the sequence. For example, $< 2, 1, 4 >$ is an occurrence of pattern C[−1,2]A[0,2]G in sequence $S =$ CACAGG. Hence sequential pattern mining with general gaps is more useful. We know that pattern matching is one of the essential tasks in pattern mining. Exact pattern matching is an ideal research, since it does not allow noise, while the approximate pattern matching can solve the problem. In conclusion, the approximate pattern matching with general gaps is a more challenging and general issue. The contributions of this paper are described specifically as follows:

(1) We propose the problem of Strict Approximate Pattern matching with general gaps and length constraints (SAP). When the Hamming distance is 0, the problem is automatically converted to the exact pattern matching, which is called the SPANGLO problem [16]. We prove that a SAP instance can be transformed into exponential SPANGLO instances; therefore, we cannot use SETS [16] to solve a SAP problem.
(2) To solve a SAP problem effectively, we propose an effective online algorithm, named SETA, which applies pruning strategies. In addition, we prove the correctness and completeness of SETA and analyse the time and space complexities of SETA, which are $O(Maxlen \times W \times m^2 \times n \times T)$ and $O(m \times Maxlen \times W \times T)$ respectively, where $m$, $Maxlen$, $W$, and $T$ are the length of pattern $P$, the maximal length constraint, the maximal span of pattern $P$ and the approximate threshold.
(3) Extensive experimental results on real biological data show the correctness of the approach of transforming a SAP instance to an exponential amount of SPANGLO instances, and also validate the correctness and effectiveness of SETA.

The rest of the paper is organized as follows. Section 2 summarizes the related work. Section 3 presents the

definition of the SAP problem and analyses the method of transforming the SAP problem into SPANGLO problems. Section 4 proposes how to create and calculate a subnet-tree for SAP and proves the correctness of the calculation method. After this, we propose the SETA algorithm and analyse the time and space complexities. Finally, we illustrate how SETA works. Section 5 validates the correctness of SETA through vast real biological data. We give the conclusion of this paper in Section 6.

## 2 Related work

Example 1 is a kind of strict pattern matching. Another kind of pattern matching is called loose pattern matching which only considers the position of the last pattern substring in the sequence. Since the last pattern substring is $p_2 = \mathrm{G}$ in Example 1, and it can match positions 4 and 5 in sequence $S$, there are only 2 occurrences in the loose pattern matching.

Because all the gaps of the pattern are no less than 0 in Example 1, bounded by the gap constraints, $p_{j+1}$ must appear in the right of $p_j$. The more general gap constraints can be negative. Thus $p_{j+1}$ can appear in either the right or the left of $p_j$. If there is a negative gap in the pattern, then the pattern is called a pattern with general gaps; when all the gaps are no less than 0, the pattern is called a pattern with non-negative gaps. Pattern matching with general gaps was applied to not only the loose pattern matching [17, 18] but also the exact pattern matching [16].

The length constraint, which is composed of the minimal length constraint and the maximal length constraint, refers to restraining the span of the occurrence, which means the distance between the minimal value and the maximal value of the occurrence. For instance, the minimal value and the maximal value of $< 2, 3, 5 >$ are 2 and 5, respectively. So the span of $< 2, 3, 5 >$ is $5 - 2 + 1 = 4$. There are 3 occurrences of pattern $P1 = \mathrm{C}[-1, 2]\mathrm{A}[0, 2]\mathrm{G}$ in sequence $S$ when the minimal and maximal length constraints are 3 and 4 respectively, i.e. $< 2, 3, 4 >$, $< 2, 1, 4 >$, and $< 2, 3, 5 >$. As the spans of $< 0, 1, 4 >$, $< 0, 3, 4 >$, and $< 0, 3, 5 >$ are 5, 5, and 6, none of them satisfies the length constraints.

What we discussed above are all examples of exact pattern matching, i.e. $p_j$ must be equal to $\mathrm{s}_i$. But in real research, most situations are approximate pattern matching. Approximate pattern matching contains pattern matching based on edit distance and on Hamming distance. In this paper, we focus on the study of approximate pattern matching with Hamming distance; namely, the distance between the pattern and the substring corresponding to the occurrence must be no greater than the given threshold. For instance, if the given threshold is 1, in Example 1, $< 0, 1, 3 >$ is an approximate occurrence with the Hamming distance being 1 because $s_3 = \mathrm{A} \neq p_2 = \mathrm{G}$. Nevertheless, $< 0, 2, 3 >$ is still not an approximate occurrence with the Hamming distance being 1, because the Hamming distance between $< 0, 2, 3 >$ and the pattern is 2. Of course, $< 0, 1, 4 >$ is also an occurrence, because the distance between $< 0, 1, 4 >$ and the pattern is 0, which is less than the threshold. We can clearly know that if the approximate threshold is 0 it will convert from approximate pattern matching to exact pattern matching. Therefore, compared with exact pattern matching, approximate pattern matching is more general.

In pattern matching with gaps, there are some special conditions. One condition is called the one-off condition, which means that any position in the sequence can be used at most once. Guo et al. [19] investigated pattern matching with the one-off condition. The one-off condition has many different names in sequential pattern mining. Actually, Ferreira and Azevedo [11], Huang et al. [20] and Lam et al. [21] all focused on pattern mining with the one-off condition. Similar to the one-off condition, Ding et al. [22] researched sequential pattern mining with the non-overlapping condition, which means that no position in the sequence can be reused by other pattern substrings. Compared with the one-off condition and the non-overlapping condition, there is no special condition which means that any position in the sequence can be used more than once. Min et al. [23] focused on pattern matching with length constraints. Zhang et al. [12] investigated sequential pattern mining with periodic gap constraints. All these researches mentioned above are with no special condition. In this article, we also focus on no special condition. Table 1 shows the related work in pattern matching.

Table 1 shows that the main difference between [16] and our work is that [16] investigated exact pattern matching, while this study addresses approximate pattern matching, which is a more general issue. Since a SAP instance can be transformed into exponential SPANGLO instances, which was handled in [16], we propose an effective algorithm, named SETA, which employs effective pruning strategies. An illustrative example is given to show how to prune effectively in Section 5.4. The following cases show the meaning of this issue.

## 3 Problem definition and analysis

### 3.1 Problem definition

**Definition 1** A **sequence** can be denoted as $S = s_0 s_1 \ldots s_i \ldots s_{n-1}$, where $n$ is the length of $S$. $\sum$ represents a set of characters, such as in the DNA sequence, where $\sum$ is {A,T,G,C}.

**Table 1** Comparison of related work

| Algorithms | Number of gaps | Gap type | Matching type | Matching type | Length constraint | Special condition |
|---|---|---|---|---|---|---|
| Manber and Baeza-Yates [5] | Single | Non-negative | Strict matching | Exact | None | No |
| Bille et al. [24] | Multiple | Non-negative | Loose matching | Exact matching | – | –[Note1] |
| Rahman et al. [25] | Multiple | Non-negative | Loose matching | Exact matching | – | – |
| Bille et al. [26] | Multiple | Non-negative | Loose /Strict matching[Note2] | Exact matching | –/None | – |
| Fredriksson and Grabowski [17, 18] | Multiple | General | Loose matching | $\delta$approximate | – | – |
| Guo et al. [19] | Multiple | Non-negative | Strict matching | Exact matching | Yes | One-off |
| He et al. [27] | Multiple | Non-negative | Strict matching | Hamming distance | Yes | One-off |
| Min et al. [23] | Multiple | Non-negative | Strict matching | Exact matching | Yes | No |
| Wu et al. [16] | Multiple | General | Strict matching | Exact matching | Yes | No |
| This paper | Multiple | General | Strict matching | Hamming distance | Yes | No |

Note 1: "–" in the table represents the items we do not take into consideration.

Note 2: This paper designed two algorithms, which research loose pattern matching and strict pattern matching respectively.

**Definition 2** A **pattern with general gaps** can be denoted as $P = p_0[a_0, b_0]p_1 \ldots [a_{j-1}, b_{j-1}]p_j \ldots [a_{m-2}, b_{m-2}]p_{m-1}$, where $m$ denotes the length of $P$, $a_j$ and $b_j$ are given integers, representing the minimal and maximal wildcards between $p_j$ and $p_{j+1}$, where $a_j \leq b_j$, and in addition, $a_j$ and $b_j$ can be negative.

**Definition 3** Given two sequences $Q = q_0q_1 \ldots q_{m-1}$ and $R = r_0r_1 \ldots r_{m-1}$, if there are $k$ positions at which the corresponding characters are different, *i.e.* $q_i \neq r_i (0 \leq i < m)$, then the **Hamming distance** between the two strings is $k (0 \leq k \leq m)$. $D(Q, R)$ is used to denote the Hamming distance between $Q$ and $R$.

**Definition 4** Given a **threshold** $d$, if a group of position indices $I =< i_0, \ldots, i_j, \ldots, i_{m-1} >$ satisfies the following equations

$$D(p_0p_1 \cdots p_{m-1}, s_{i_0}s_{i_1} \cdots s_{i_{m-1}}) \leq d \tag{1}$$

$$i_{j-1} \neq i_j \tag{2}$$

$$\begin{cases} \min_{j-1} \leq i_j - i_{j-1} - 1 \leq \max_{j-1}, \text{if } i_{j-1} < i_j \\ \min_{j-1} \leq i_j - i_{j-1} \leq \max_{j-1}, \text{if } i_{j-1} > i_j \end{cases} \tag{3}$$

where $0 \leq j \leq m-1$ and $0 \leq i_j \leq n-1$, then $I$ is an **approximate occurrence** of $P$ in $S$.

**Definition 5** An approximate occurrence $I$ satisfies the **length constraint** which means that the occurrence is subject to the following equation

$$Minlen \leq i_{max} - i_{min} + 1 \leq Maxlen \tag{4}$$

In addition, $i_{max} - i_{min} + 1$ is the **span** of occurrence $I$, where $i_{max} = \max(i_0, \ldots, i_j, \ldots, i_{m-1})$, $i_{min} = \min(i_0, \ldots, i_j, \ldots, i_{m-1})$, and *Minlen* and *Maxlen* are the two given integers which are the **minimal and maximal length constraint**, respectively.

**Definition 6** Let the set $T(S, P, d)$ denote all the approximate occurrences and $|T(S, P, d)|$ denote the length of $T(S, P, d)$. SAP is to calculate $|T(S, P, d)|$.

### 3.2 Theoretical analysis

From Table 1, we know that SPANGLO [16] handles the exact matching problem, while SAP deals with the approximate matching problem. Given sequence $S$, pattern $P$, *Minlen* and *Maxlen*, apparently, the SPANGLO problem can be denoted by $|T(S, P, 0)|$.

**Theorem 1** *A SAP instance can be transformed into exponential SPANGLO instances.*

*Proof* Let $f(S, P, k) = |T(S, P, k)| - |T(S, P, k-1)|$. We can know that $f(S, P, k)$ denotes the number of occurrences whose Hamming distance between the approximate occurrence and the pattern is $k$. That is to say, arbitrarily choose $k$ different positions in pattern $P$ to make the corresponding character differ from $p_j$. So there are $C_m^k = \frac{m!}{k!*(m-k)!}$ different choices. There are $|\sum|$ -1 different choices in each different position. Therefore, $f(S, P, k)$ can be transformed into $C_m^k * (|\Sigma| - 1)$ SPANGLO instances. Since $|T(S, P, d)|$ is $\sum_{i=0}^{d} f(S, P, i)$, $|T(S, P, d)|$ can be transformed into $1 + \sum_{i=1}^{d} C_m^i * (|\Sigma| - 1)$ SPANGLO instances. Hence **Theorem 1** is proved.

Wu et al. [16] proposed an effective algorithm, named SETS, to solve SPANGLO. From Theorem 1, we can know that SETS fails to solve SAP, since a SAP instance

will be transformed into exponential SPANGLO instances. Therefore, we have to propose a new algorithm to solve SAP. □

## 4 Subnettree for SAP

### 4.1 Subnettree

**Definition 7** A **Nettree** [28] is an extension of a tree, because it has many concepts similar to a tree, such as the root, leaf, level, parent, child and so on. Nettree has four features which are obviously different from a tree.

(1) A Nettree may have $n$ roots, where $n \geq 1$;
(2) Some nodes other than roots in a Nettree may have many parents;
(2) There may be more than 1 path from a node to its ancestor node in a Nettree;
(4) The same node label can appear in different levels in a Nettree. $n^i_j$ denotes the node $i$ in the $j$-th level.

To solve SAP, a subnettree is also employed since we can confirm the maximal value in the subnettree. So we can deal with the length constraint. More important is that, through this approach, an online algorithm is proposed.

**Definition 8** A **subnettree** [16] is composed of three parts, a **central node** $n^i_j$, its **ancestor nodes** $A(n^i_j)$, and its **descendant nodes** $D(n^i_j)$, where the ancestor node refers to the fact that node $n^c_b$ is on the path from node $n^i_j$ to a root, $c \leq i$, and $1 \leq b < j$. Similarly, the descendant node refers to the fact that node $n^e_f$ is on the path from node $n^i_j$ to a leaf, and $e < i$. Subnettree $n^i_j$ is used to represent a subnettree with a central node $n^i_j$.

From Definition 8, we see that there is only one node $n^i_j$ in the $j$-th level, and $i$ is the maximal node label in the subnettree, the maximal ancestor node label of $n^i_j$ can be $i$ and the maximal descendant node label can just be $i$-1.

**Lemma 1** *When we create subnettree $n^i_j$ according to pattern $P$, in the following three cases, the subnettree can be omitted.*

Case 1. *$j$ is equal to $m$, and gap $b_{m-1}$ is less than 0.*
Case 2. *$j$ is equal to 1, and gap $a_0$ is no less than 0.*
Case 3. *gap $b_{j-2}$ is less than 0 or gap $a_{j-1}$ is no less than 0, where $1 < j < m$.*

*Proof* Case 1. When $j$ is equal to $m$, it indicates that $p_{m-1}$ matches with $s_i$. Since $a_{m-1}$ is less than $b_{m-1}$, if $b_{m-1}$ is less than 0, then $a_{m-1}$ is also less than 0. Suppose character $s_h$ match with $p_{m-2}$, $h$ is greater than $i$ according to gap $[a_{m-1}, b_{m-1}]$. This is contradictory to the definition of the subnettree.

Case 2. When $j$ is equal to 1, it indicates that $p_0$ matches with $s_i$. If $a_0$ is no less than 0, then $b_0$ is also no less than 0. Suppose $s_h$ match with $p_1$, $h$is greater than $i$ according to gap $[a_0, b_0]$, which is contradictory to the definition of the subnettree.

Case 3. Similarly, if gap $b_{j-2}$ is less than 0 or gap $a_{j-1}$ is no less than 0 ($1 < j < m$), then position $h$ is greater than $i$ according to the gap, where $h$ is the position of character $s_h$, which will match $p_{j-2}$ or $p_j$. This is contradictory to the definition of the subnettree. Therefore the lemma is proved.

To confirm the range of the node labels in the $j$-th level of the subnettree, we propose the definitions of the maximal sibling and the minimal sibling in the $j$-th level. □

**Definition 9** The **minimal sibling** and the **maximal sibling** are the minimal and maximal node labels in the $k$-th level of the subnettree and denoted by $c_k$ and $e_k$, respectively.

**Lemma 2** *In the process of creating the ancestor nodes of subnettree $n^i_j$, we create the nodes in the k-th level according to the nodes in the k+1-th level, where $1 \leq k < j$. In this process, $c_k$ and $e_k$ can be calculated by (5) and (6), respectively.*

*Similarly, in the process of creating the descendant nodes of subnettree $n^i_j$, we create the nodes in the k-th level according to the nodes in the k-1-th level, where $j < k \leq m$. In this process, $c_k$ and $e_k$ can be calculated by (7) and (8), respectively.*

$$c_k = \begin{cases} \max(0, i - Maxlen + 1, c_{k+1} - b_{k-1} - 1) & b_{k-1} \geq 0 \\ \max(0, i - Maxlen + 1, c_{k+1} - b_{k-1}) & b_{k-1} < 0 \end{cases} \tag{5}$$

$$e_k = \begin{cases} \min(i, e_{k+1} - a_{k-1} - 1) & a_{k-1} \geq 0 \\ \min(i, e_{k+1} - a_{k-1}) & a_{k-1} < 0 \end{cases} \tag{6}$$

$$c_k = \begin{cases} \max(0, i - Maxlen + 1, c_{k-1} + a_{k-2} + 1) & a_{k-2} \geq 0 \\ \max(0, i - Maxlen + 1, c_{k-1} + a_{k-2}) & a_{k-2} < 0 \end{cases} \tag{7}$$

$$e_k = \begin{cases} \min(i - 1, e_{k-1} - b_{k-2} + 1) & b_{k-2} \geq 0 \\ \min(i - 1, e_{k-1} - b_{k-2}) & b_{k-2} < 0 \end{cases} \tag{8}$$

*Proof* First of all, we prove the method of calculating $c_k$ and $e_k$ in the ancestor set. Obviously, the minimal sibling $c_k$ is no less than 0. Since $c_k$ needs to satisfy the length constraint, $c_k$ is also no less than $i - Maxlen + 1$. Besides, since the nodes in the $k$-th level and $k$+1-th level correspond with $p_{k-1}$ and $p_k$ respectively, $c_k$ also needs to satisfy gap $b_{k-1}$.

According to (2), when $b_{k-1}$ is less than 0, $c_k$ is no less than $c_{k+1} - b_{k-1}$, while $b_{k-1}$ is no less than 0, $c_k$ is no less than $c_{k+1} - b_{k-1} - 1$. Therefore, $c_k$ can be calculated by (5). Similarly, according to Definition 8, the maximal value of $e_k$ in subnettree $n_j^i$ is $i$ and $e_k$ also needs to satisfy gap $a_{k-1}$. Hence, $e_k$ can be calculated by (6).

In the descendant set, the method of calculating $c_k$ and $e_k$ is similar to that in the ancestor set. The difference is that in the descendant set, the nodes in the $k$-th level are created according to the nodes in the $k$-1-th level. So the values of $c_k$ and $e_k$ are calculated according to $c_{k-1}$ and $e_{k-1}$ respectively. Both of them need to satisfy gap $a_{k-2}$ and $b_{k-2}$ respectively. Therefore, $c_k$ can be calculated by (7). Since the maximal value of the descendant nodes in the subnettree is $i - 1$, $e_k$ can be calculated by (8). Hence Lemma 2 is proved.

Since there are some paths that can satisfy the length constraint and others fail to do so, we propose several concepts to distinguish the two kinds of path. □

**Definition 10** Let $M$ be a path from node $n_j^i$ to node $n_b^c$, where $0 \leq i < n$, $1 \leq j$, $0 \leq c < n$, and $1 \leq b.e$ is the minimal node label in this path, i.e. $e = \min(M)$, if path $M$ satisfies the length constraint, i.e. $Minlen \leq i - e + 1 \leq Maxlen$, then we say that $M$ is a **path with length constraint**; otherwise, $M$ is a **complement path with length constraint**.

**Definition 11 NAPS (Number of Ancestor Paths with Similarity constraint)** is the number of paths which are from an ancestor node $n_k^l$ to its central node $n_j^i$ with the Hamming distance $d$, denoted by $N_A(n_j^i, n_k^l, d)$. In these paths, the number of paths that satisfy the length constraint is called **NAPLC (Number of Ancestor Paths with Length Constraints)**, denoted by $N_A^C(n_j^i, n_k^l, d)$ and the number of paths that do not satisfy the length constraint is called **NCAPLC (Number of Complement of Ancestor Paths with Length Constraints)**, denoted by $N_A^{\sim}(n_j^i, n_k^l, d)$. The initial value of $N_A(n_j^i, n_j^i, d)$ is set as follows: if $s_i = p_{j-1}$, then the distance between $s_i$ and $p_{j-1}$ is 0, or else is 1. Therefore, if $s_i = p_{j-1}$, then $N_A(n_j^i, n_j^i, 0)$=1 and for any $d > 0$, $N_A(n_j^i, n_j^i, d)$ is 0. Otherwise, $N_A(n_j^i, n_j^i, 1) = 1$, and for any $d \neq 1$, $N_A(n_j^i, n_j^i, d)$ is 0.

Obviously, $N_A(n_j^i, n_k^l, d)$ is the sum of $N_A^C(n_j^i, n_k^l, d)$ and $N_A^{\sim}(n_j^i, n_k^l, d)$, *i.e.*$N_A(n_j^i, n_k^l, d) = N_A^C(n_j^i, n_k^l, d) + N_A^{\sim}(n_j^i, n_k^l, d)$. Next, we will show how to calculate $N_A(n_j^i, n_k^l, d)$, $N_A^C(n_j^i, n_k^l, d)$, and $N_A^{\sim}(n_j^i, n_k^l, d)$.

**Lemma 3** *$N_A(n_j^i, n_k^l, d)$ is calculated according to (9).*

*Proof* If $s_l = p_k$, then the distance between $s_l$ and $p_k$ is 0, or else is 1. We know that the nodes in the $k$-th level are created according to the nodes in the $k$+1-th level in the ancestor nodes of the subnettree. Therefore, after adding node $n_k^l$, if $s_l = p_k$, then the distance is not changed, $N_A(n_j^i, n_k^l, d)$ is the sum of $N_A(n_j^i, n_{k+1}^{r_q}, d)$; otherwise, the distance increases by 1, hence $N_A(n_j^i, n_k^l, 0) = 0$ and $N_A(n_j^i, n_k^l, d)$ is the sum of $N_A(n_j^i, n_{k+1}^{r_q}, d\text{-}1)$, where $n_{k+1}^{r_q}$ is the $q$th child node of $n_k^l$. Therefore Lemma 3 is proved. □

**Lemma 4** *$N_A^C(n_j^i, n_k^l, d)$ and $N_A^{\sim}(n_j^i, n_k^l, d)$ can be calculated according to (10) and (11), respectively.*

*Proof* If the distance between $l$ and $i$ satisfies the length constraint, then $N_A^C(n_j^i, n_k^l, d) = N_A(n_j^i, n_k^l, d)$ and $N_A^{\sim}(n_j^i, n_k^l, d) = 0$. Otherwise, we need to consider whether $s_l$ is the same as $p_k$ or not, if $s_l$ and $p_k$ are the same, $N_A^C(n_j^i, n_k^l, d)$ and $N_A^{\sim}(n_j^i, n_k^l, d)$ is the sum of $N_A^C(n_j^i, n_{k+1}^{r_q}, d)$ and $N_A^{\sim}(n_j^i, n_{k+1}^{r_q}, d)$, respectively. If $s_l$ and $p_k$ are different, then $N_A^C(n_j^i, n_k^l, 0)$ and $N_A^{\sim}(n_j^i, n_k^l, 0)$ are 0, besides, $N_A^C(n_j^i, n_k^l, d)$ and $N_A^{\sim}(n_j^i, n_k^l, d)$ should be the sum of $N_A^C(n_j^i, n_{k+1}^{r_q}, d\text{-}1)$ and $N_A^{\sim}(n_j^i, n_{k+1}^{r_q}, d\text{-}1)$, respectively. Hence Lemma 4 is proved. □

$$N_A(n_j^i, n_k^l, d) = \begin{cases} \sum_{q=1}^{t} N_A(n_j^i, n_{k+1}^{r_q}, d) & \\ \quad s_l = p_k & \\ \sum_{q=1}^{t} N_A(n_j^i, n_{k+1}^{r_q}, d-1) & \\ \quad s_l \neq p_k \text{ and } d > 0 & \\ 0 \quad s_l \neq p_k \text{ and } d = 0 & \end{cases} \tag{9}$$

where $n_{k+1}^{r_q}$ and $t$ represent the $q$-th child node of $n_k^l$ and the number of child nodes of the node $n_k^l$ in the subnettree, respectively.

$$N_A^C(n_j^i, n_k^l, d) = \begin{cases} N_A\left(n_j^i, n_k^l, d\right) & Minlen \leq i - l + 1 \leq Maxlen \\ \sum_{q=1}^{t} N_A^C\left(n_j^i, n_{k+1}^{r_q}, d\right) & s_l = p_k \text{ and } (i - l + 1 > Maxlen \text{ or } i - l + 1 < Minlen) \\ \sum_{q=1}^{t} N_A^C\left(n_j^i, n_{k+1}^{r_q}, d-1\right) & s_l \neq p_k \text{ and } (i - l + 1 > Maxlen \text{ or } i - l + 1 < Minlen) \text{ and } d > 0 \\ 0 & s_l \neq p_k \text{ and } (i - l + 1 > Maxlen \text{ or } i - l + 1 < Minlen) \text{ and } d = 0 \end{cases} \tag{10}$$

where $n_{k+1}^{r_q}$and $t$ represent the $q$-th child node of $n_k^l$ and the number of child nodes of the node $n_k^l$ in the subnettree respectively.

$$N_A^{\sim}(n_j^i, n_k^l, d) = \begin{cases} 0 & Minlen \le i - l + 1 \le Maxlen \\ \sum_{q=1}^{t} N_A^{\sim}\left(n_j^i, n_{k+1}^{r_q}, d\right) & s_l = p_k \text{ and } (i - l + 1 > Maxlen \text{ or } i - l + 1 < Minlen) \\ \sum_{q=1}^{t} N_A^{\sim}\left(n_j^i, n_{k+1}^{r_q}, d-1\right) & s_l \neq p_k \text{ and } (i - l + 1 > Maxlen \text{ or } i - l + 1 < Minlen) \text{ and } d > 0 \\ 0 & s_l \neq p_k \text{ and } (i - l + 1 > Maxlen \text{ or } i - l + 1 < Minlen) \text{ and } d = 0 \end{cases} \tag{11}$$

where $n_{k+1}^{r_q}$ and $t$ represent the $q$-th child node of $n_k^l$ and the number of child nodes of the node $n_k^l$ in the subnettree respectively.

**Definition 12 NRPLC (Number of Root Paths with Length Constraint)** is the number of paths which are from roots to the central node $n_j^i$ with the Hamming distance $d$ which satisfy the length constraint, denoted by $N_R^C(n_j^i, d)$. Similarly, the number of paths which do not satisfy the length constraint is called **NCRPLC (Number of Complement of Root Paths with Length Constraint)**, denoted as $N_R^{\sim}(n_j^i, d)$.$N_R^C(n_j^i, d)$ and $N_R^{\sim}(n_j^i, d)$ can be calculated by the following equations.

$$N_R^C(n_j^i, d) = \sum_{q=1}^{t} N_A^C(n_j^i, n_1^{r_q}, d) \tag{12}$$

$$N_R^{\sim}(n_j^i, d) = \sum_{q=1}^{t} N_A^{\sim}(n_j^i, n_1^{r_q}, d) \tag{13}$$

where $n_1^{r_q}$and $t$ represent the $q$-th rootand the number of roots in the subnettree, respectively.

Now, we will introduce how to create the descendant nodes of central node $n_j^i$.

**Definition 13 NDPS (Number of Descent Paths with Similarity constraint)** is the number of paths which are from central node $n_j^i$ to its descendant node $n_k^l$ with the Hamming distance $d$, denoted by $N_D(n_j^i, n_k^l, d)$. In these paths, the number of paths that satisfy the length constraint is called **NDPLC (Number of Descent Paths with Length Constraints)**, denoted by $N_D^C(n_j^i, n_k^l, d)$ and the number of paths that do not satisfy the length constraint is called **NCDPLC (Number of Complement of Descent Paths with Length Constraints)**, denoted by $N_D^{\sim}(n_j^i, n_k^l, d)$. When we initialize NDPS, it is different from the method of initializing NAPS. In this definition, no matter whether $s_i$ is the same as $p_{j-1}$ or not, $N_D(n_j^i, n_j^i, 0)$=1 and for any $d >$ 0, $N_D(n_j^i, n_j^i, d)$ =0. The reason is that when $s_i$ and $p_{j-1}$ are different, if $N_D(n_j^i, n_j^i, 1)$ =1, then it will cause the fact that $s_i$ and $p_{j-1}$ are different to be calculated excessively.

**Lemma 5** *$N_D(n_j^i, n_k^l, d)$, $N_D^C(n_j^i, n_k^l, d)$, and $N_D^{\sim}(n_j^i, n_k^l, d)$ can be calculated by (14), (15), and (16), respectively.*

*Proof* Similar to the method of calculating NAPS, NAPLC and NCAPLC of the ancestor nodes, we know that the nodes in the $k$-th level are created according to the nodes in the $k$-1-th level nodes in the descendant nodes of the subnettree. After adding node $n_k^l$, if $s_l = p_k$, then the distance is not changed, $N_D(n_j^i, n_k^l, d)$ is the sum of $N_D(n_j^i, n_{k-1}^{r_q}, d)$, otherwise, the distance increases by 1, $N_D(n_j^i, n_k^l, 0)$=0, and $N_D(n_j^i, n_k^l, d)$ is the sum of $N_D(n_j^i, n_{k-1}^{r_q}, d-1)$, where $n_{k-1}^{r_q}$ is the $q$-th parent node of node $n_k^l$. Hence, $N_D(n_j^i, n_k^l, d)$ can be calculated according to (14).

If $l$ and $i$ satisfy the length constraint, the values of NDPLC and NCDPLC of $n_k^l$ are the value of NDPS of $n_k^l$ and 0, respectively. Otherwise, we need to judge whether $s_l$ and $p_k$ are the same or not. If $s_l$ and $p_k$ are the same, $N_D^C(n_j^i, n_k^l, d)$ and $N_D^{\sim}(n_j^i, n_k^l, d)$ are the sum of $N_D^C(n_j^i, n_{k-1}^{r_q}, d)$ and $N_D^{\sim}(n_j^i, n_{k-1}^{r_q}, d)$, respectively; otherwise, both $N_D^C(n_j^i, n_k^l, 0)$ and $N_D^{\sim}(n_j^i, n_k^l, 0)$ are 0, and $N_D^C(n_j^i, n_k^l, d)$ and $N_D^{\sim}(n_j^i, n_k^l, d)$ are the sum of $N_D^C(n_j^i, n_{k-1}^{r_q}, d-1)$ and $N_D^{\sim}(n_j^i, n_{k-1}^{r_q}, d-1)$, respectively. Therefore **Lemma 5** is proved. □

$$N_D(n_j^i, n_k^l, d) = \begin{cases} \sum_{q=1}^{t} N_D(n_j^i, n_{k-1}^{r_q}, d) & s_l = p_k \\ \sum_{q=1}^{t} N_D(n_j^i, n_{k-1}^{r_q}, d-1) & s_l \neq p_k \text{ and } d > 0 \\ 0 & s_l \neq p_k \text{ and } d = 0 \end{cases} \tag{14}$$

where $n_{k-1}^{r_q}$ and $t$ represent the $q$-th parent node of node $n_k^l$ and the number of parent nodes of $n_k^l$ in the subnettree, respectively.

$$N_D^C(n_j^i, n_k^l, d) = \begin{cases} N_D\left(n_j^i, n_k^l, d\right) & Minlin \le i-l+1 \le Maxlin \\ \sum_{q=1}^{t} N_D^C\left(n_j^i, n_{k-1}^{r_q}, d\right) & s_l = p_k \text{ and } (i-l+1 > Maxlen \text{ or } i-l+1 < Minlen) \\ \sum_{q=1}^{t} N_D^C\left(n_j^i, n_{k-1}^{r_q}, d-1\right) & s_l \neq p_k \text{ and } (i-l+1 > Maxlen \text{ or } i-l+1 < Minlen) \text{ and } d > 0 \\ 0 & s_l \neq p_k \text{ and } (i-l+1 > Maxlen \text{ or } i-l+1 < Minlen) \text{ and } d = 0 \end{cases} \tag{15}$$

where $n_{k-1}^{r_q}$ and $t$ represent the $q$-th parent node of node $n_k^l$ and the number of parent nodes of $n_k^l$ in the subnettree, respectively.

$$N_D^{\sim}(n_j^i, n_k^l, d) = \begin{cases} 0 & Minlen \le i-l+1 \le Maxlen \\ \sum_{q=1}^{t} N_D^{\sim}(n_j^i, n_{k-1}^{r_q}, d) & s_l = p_k \text{ and } (i-l+1 > Maxlen \text{ or } i-l+1 < Minlen) \\ \sum_{q=1}^{t} N_D^{\sim}(n_j^i, n_{k-1}^{r_q}, d-1) & s_l \neq p_k \text{ and } (i-l+1 > Maxlen \text{ or } i-l+1 < Minlen) \text{ and } d > 0 \\ 0 & s_l \neq p_k \text{ and } (i-l+1 > Maxlen \text{ or } i-l+1 < Minlen) \text{ and } d = 0 \end{cases} \tag{16}$$

where $n_{k-1}^{r_q}$ and $t$ represent the $q$-th parent node of node $n_k^l$ and the number of parent nodes of $n_k^l$ in the subnettree, respectively.

**Lemma 6** *If $\sum_{d=0}^{T} N_A(n_j^i, n_k^l, d) = 0$, then node $n_k^l$ has no parent. Similarly, if $\sum_{d=0}^{T} N_D(n_j^i, n_k^l, d) = 0$, then node $n_k^l$ has no child.*

*Proof* Suppose $n_g^f$ is an ancestor node of $n_k^l$. We know that the Hamming distance of each path from $n_g^f$ to $n_j^i$ is no less than that of the path from $n_k^l$ to $n_j^i$. If $\sum_{d=0}^{T} N_A(n_j^i, n_k^l, d) = 0$, it indicates that in the subnettree, the Hamming distance of each path from $n_k^l$ to $n_j^i$ is greater than $d$. Therefore, the Hamming distance of each path from $n_g^f$ to $n_j^i$ is also greater than $d$. Hence, we do not need to calculate the parent of the ancestor of node $n_k^l$. Similarly, in the process of creating descendant nodes, if $\sum_{d=0}^{T} N_D(n_j^i, n_k^l, d) = 0$, we can safely prune the sub-tree whose root is node $n_k^l$. Hence, **Lemma 6** is proved. □

**Definition 14** Similar to the concepts of NRPLC and NCRPLC, **NLPLC (Number of Leaf Paths with Length Constraint)** is the number of paths which are from central node $n_j^i$ to the leaves in the $m$-th level with the Hamming distance $d$ and which satisfy the length constraint, denoted by $N_L^C(n_j^i, d)$, while **NCLPLC (Number of Complement of Leaf Paths with Length Constraint)** is the number of paths which do not satisfy the length constraint, denoted by $N_L^{\sim}(n_j^i, d)$. $N_L^C(n_j^i, d)$ and $N_L^{\sim}(n_j^i, d)$ can be calculated by the following equations.

$$N_L^C(n_j^i, d) = \sum\nolimits_{q=1}^{t} N_D^C(n_j^i, n_m^{r_q}, d) \tag{17}$$

$$N_L^{\sim}(n_j^i, d) = \sum\nolimits_{q=1}^{t} N_D^{\sim}(n_j^i, n_m^{r_q}, d) \tag{18}$$

where $n_m^{r_q}$ and $t$ represent the $q$-th leaf and the number of leaves in the subnettree, respectively.

**Definition 15** Suppose $M$ is a path from a root to a leaf which passes through node $n_j^i$, and besides, the Hamming distance between the corresponding string of this path in sequence $S$ and $p_0 p_1 \ldots p_{m-1}$ is less than the approximate threshold $d$, we can say that $M$ is a root-leaf path with length and similarity constraints or a **root-leaf path** for short. We use $N_T^C(n_j^i, d)$ to denote the number of root-leaf paths.

**Lemma 7** *$N_T^C(n_j^i, d)$ can be calculated as follows.*

$$N_T^C(n_j^i, d) = \sum_{k=0}^{d}\sum_{e=0}^{d-k}(N_R^C(n_j^i, k)^* N_L^{\sim}(n_j^i, e) + N_R^{\sim}(n_j^i, k)^* N_L^C(n_j^i, e) + N_R^C(n_j^i, k)^* N_L^C(n_j^i, e)) \quad (19)$$

*Proof* A root-leaf path in subnettree $n_j^i$ satisfies one of the following three cases.

Case 1. The sub-path from a root to central node $n_j^i$ satisfies the length constraint and the other sub-path from $n_j^i$ to a leaf fails to satisfy the length constraint. We can know that there are $N_R^C(n_j^i, k, Minlen, Maxlen)^* N_L^{\sim}(n_j^i, e, Minlen, Maxlen)$ root-leaf paths in this case.

Case 2. The sub-path from a root to $n_j^i$ does not satisfy the length constraint, but the other sub-path from $n_j^i$ to a leaf satisfies the length constraint. Similarly, there are $N_R^{\sim}(n_j^i, k, Minlen, Maxlen)^* N_L^C(n_j^i, e, Minlen, Maxlen)$ root-leaf paths in this case.

Case 3. Both of the sub-paths satisfy the length constraint. There are $N_R^C(n_j^i, k, Minlen, Maxlen)^* N_L^C(n_j^i, e, Minlen, Maxlen)$ root-leaf paths in this case.

Since the sum of $k$ and $e$ is no greater than the approximate threshold $d$, therefore, $N_T^C(n_j^i, d, Minlen, Maxlen)$ can be calculated according to (19). □

**Theorem 1** *$|T(S, P, d)|$ can be calculated as follows.*

$$|T(S, P, d)| = \sum_{i=Minlen-1}^{n-1}\sum_{j=1}^{m} N_T^C(n_j^i, d) \quad (20)$$

*where n, m, d, and Minlen are the lengths of the sequence and the pattern, the approximate constraint and the minimal length constraint, respectively.*

*Proof* Since all the subnettrees have $m$ levels, $j$ can vary from 1 to $m$. Since $i$ is the maximal value in subnettree $n_j^i$, $i$ must be no less than *Minlen*-1 according to Definition 5. We see that $i$ is also no greater than $n$-1. Each subnettree has $N_T^C(n_j^i, d)$ approximate occurrences. Hence, $|T(S, P, d)|$ can be calculated according to (20). Therefore Theorem 1 is proved. □

## 4.2 SETA

We give the SETA algorithm as follows:

SETA algorithm

Input: $P=p_0[a_0,b_0]p_1 \cdots [a_{j-1},b_{j-1}]p_j \cdots [a_{m-2},b_{m-2}]p_{m-1}$, $S = s_0 s_1 \cdots s_{n-1}$, *Minlen*, *Maxlen*, and threshold $d$

Output: $|T(S,P,d)|$

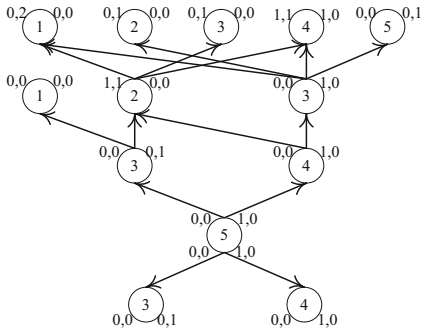
```
1: sum=0;
2: for i=Minlen-1 to n-1 step 1
3:     for j=m-1 down to 0 step -1
4:         if subnettree n^i_j can be created according to Lemma 1 then
5:             Create central node n^i_j and initialize NAPS and NDPS
6:             for k=j-1 down to 0 step -1
7:                 Calculate c_k and e_k according to equations (5) and (6)
8:                 for t=c_k to e_k step 1
9:                     if this satisfies Lemma 6 then continue
10:                        Create node n^t_k and calculate its NAPS, NAPLC, and NCAPLC according to equations (9), (10), and (11), respectively
11:                    next t
12:                next k
13:                Calculate NRPLC and NCRPLC according equations (12) and (13), respectively
14:                for k=j+1 to m-1 step 1
15:                Calculate c_k and e_k according to equations (7) and (8), respectively
16:                for t=c_k to e_k step 1
17:                    if this satisfies Lemma 6 then continue
18:                    Create node n^t_k and calculate its NDPS, NDPLC and NCDPLC according to equations (14), (15), and (16), respectively
19:                next t
20:            next k
21:            calculate NLPLC and NCLPLC according to equations (17) and (18), respectively
22:            calculate N^C_T(n^i_j,d) according to equation (19), and sum+=N^C_T(n^i_j,d)
23:        end if
24:    next j
25: next i
26: return sum;
```

## 4.3 Analysis

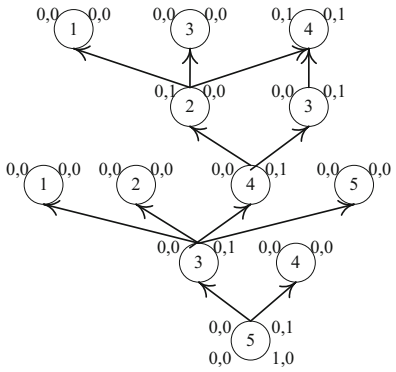
Apparently, according to Theorem 1, we know that SETA is a complete algorithm. Next, we will analyse the space and time complexities of SETA.

**Theorem 2** *The space and time complexities of SETA are $O(m \times Maxlen \times W \times d)$ and $O(Maxlen \times W \times m^2 \times n \times d)$, respectively, where m, Maxlen, W, and d are the length of pattern P, the maximal length constraint, the maximal gap of pattern P and the approximation threshold, respectively.*

*Proof* It is easy to know that the space complexity of SETA is $O(m \times Maxlen \times W \times d)$, since there are at most $m$ levels in a subnettree, each level has no more than *Maxlen* nodes, and each node has at most $W$ parent (or child) nodes, *i.e.* $W = \max(max_j - min_j + 1)(0 \le j \le m-1)$. Besides, there



(A) Subnettree $n^5_4$



(B) Subnettree $n^5_5$

**Fig. 2** Some subnettrees

are $d+1$ groups of space in each node, which store NAPS, NAPLC, and NCAPLC (or NDPS, NDPLC, and NCDPLC) in the distance from 0 to $d$, respectively.

The time complexities of line 7 and line 15 are both $O(1)$. Since each node has no more than $W$ parent (or child) nodes and $d+1$ groups NAPS, NAPLC, and NCAPLC (or NDPS, NDPLC, and NCDPLC), the time complexities of line 10 and line 18 are both $O(W \times d)$. There are at most *Maxlen* nodes in each level, and the times of the loop in line 8 and line 16 at most are $O(Maxlen)$. Similarly, the time complexities of line 13 and line 21 are both $O(Maxlen)$. The time complexity of line 22 is $O(d^2)$, in general $d \ll m$, so $d^2$ is neglected. In conclusion, the time complexity from line 4 to line 23 is $O(Maxlen \times W \times m \times d)$. Therefore, the time complexity of SETA is $O(Maxlen \times W \times m^2 \times n \times d)$. □

### 4.4 A running example

An illustrative instance is used to show how the SETA algorithm works.

*Example 2* Given sequence $S$ = ATGGAGAGA, pattern $P$ = A[-2,1]G[0,1]A[-2,1]G[-2,1]A, *Minlen*= 4 and *Maxlen*= 5, and threshold d = 1, we calculate $|T(S, P, d, Minlen, Maxlen)|$.

Suppose $i$ and $j$ are 5 and 3 in lines 2 and 3, respectively. Since $b_2$ =1 is greater than 0 and $a_3$ =-2 is less than 0, according to line 4, we will create subnettree $n^5_4$ shown in Fig. 2(A). The left-top, right-top, left-bottom, and right-bottom values are the node's NAPLC, NCAPLC, NDPLC, and NCDPLC, respectively. Since threshold $d$ is 1, each group has $d+1$ =2 integers. The first one is the corresponding value with distance 0, while the last one is the value with distance 1.

Since there are many nodes in the subnettree and the calculating equations for each node are the same, we choose only one node in each level to illustrate how to calculate its NAPLC, NCAPLC, NDPLC, and NCDPLC, and the other nodes are ignored. Since $s_5 = p_3$ ="G", according to SETA, we know that $N^C_A(n^5_4, n^5_4, 0, LEN) = 1$, $N^C_A(n^5_4, n^5_4, 1, LEN) = 0$, $N^{\sim}_A(n^5_4, n^5_4, 0, LEN) = N^{\sim}_A = 0$, and $N^C_D(n^5_4, n^5_4, 0, LEN) = 1$, $N^C_D(n^5_4, n^5_4, 1, LEN) = 0$, $N^{\sim}_D(n^5_4, n^5_4, 0, LEN) = N^{\sim}_D(n^5_4, n^5_4, 1, LEN)$ =0. Now we will create the ancestor nodes of $n^5_4$. Since $s_4 = p_2$ ="A" and $5 - 4 + 1 = 2$ is less than *Minlen*, according to line 10, $N^C_A(n^5_4, n^4_3, 0, LEN) = N^C_A(n^5_4, n^4_3, 1, LEN) = 0$, $N^{\sim}_A(n^5_4, n^4_3, 0, LEN)$ =1, and $N^{\sim}_A(n^5_4, n^4_3, 1, LEN) = 0$. Because node $n^2_2$ has 2 child nodes, $n^3_3$ and $n^4_3$, and *Minlen*= $4 \le 5 - 2 + 1 = 4 \le$ *Maxlen* = 5, therefore, $N^C_A(n^5_4, n^2_2, 0, LEN) = 1$, $N^C_A(n^5_4, n^2_2, 1, LEN) = 1$, $N^{\sim}_A(n^5_4, n^2_2, 0, LEN) = N^{\sim}_A(n^5_4, n^2_2, 1, LEN)$ =0. Similarly, $N^C_A(n^5_4, n^3_1, 0, LEN) = 0$, $N^C_A(n^5_4, n^3_1, 1, LEN) = 1$, $N^{\sim}_A(n^5_4, n^3_1, 0, LEN) = N^{\sim}_A(n^5_4, n^3_1, 1, LEN) = 0$. Now we will create the descendant nodes of $n^5_4$. According to lines 14 to 20, we know that $N^C_D(n^5_4, n^4_5, 0, LEN) = 1$, $N^C_D(n^5_4, n^4_5, 1, LEN) = 0$, and $N^{\sim}_D(n^5_4, n^4_5, 0, LEN) = N^{\sim}_D(n^5_4, n^4_5, 1, LEN) = 0$. Hence, according to line 22, we know that $N^C_T$ of subnettree $n^5_4$ is $2+1+1+2+1=7$.

Subnettree $n^5_5$ is shown in Figure 2(B) which is used to illustrate that SETA is an effective algorithm, since some nodes, such as $n^1_2$, $n^4_2$, and $n^5_2$, are pruned according to line 9 and 17. Besides, since sub-pattern "G[0, 1]A" satisfies Lemma 1, we also do not create subnettree $n^5_2$.

**Table 2** Real biological sequences

| Sequence | From | Length |
|---|---|---|
| *S*1 | Homo Sapiens AX829174 | 2500 |
| *S*2 | Homo Sapiens AX829174 | 5000 |
| *S*3 | Homo Sapiens AX829174 | 10011 |
| *S*4 | Homo Sapiens AL158070 | 40000 |
| *S*5 | Homo Sapiens AL158070 | 80000 |
| *S*6 | Homo Sapiens AL158070 | 167005 |
| *S*7 | Homo Sapiens AB038490 | 30000 |
| *S*8 | Homo Sapiens AB038490 | 60000 |
| *S*9 | Homo Sapiens AB038490 | 131892 |

**Table 3** Results of SETS and SETA

| | SETS | | | | | | | | | | | | | SET A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sequence | $P1$ | $Q1$ | $Q2$ | $Q3$ | $Q4$ | $Q5$ | $Q6$ | $Q7$ | $Q8$ | $Q9$ | $Q10$ | $Q11$ | $Q12$ | $P1,d$=1 |
| $S1$ | 1024 | 889 | 1113 | 1492 | 1479 | 1013 | 1396 | 1949 | 1085 | 1295 | 1770 | 1412 | 986 | 16903 |
| $S2$ | 2046 | 1740 | 2291 | 2677 | 3318 | 2240 | 2863 | 3971 | 2397 | 2523 | 3461 | 2841 | 2043 | 34411 |
| $S3$ | 3890 | 2815 | 4081 | 5172 | 6905 | 4311 | 7083 | 8321 | 4224 | 6113 | 5821 | 4396 | 3186 | 66318 |
| $S4$ | 15999 | 10082 | 14129 | 21236 | 29681 | 19210 | 37664 | 35112 | 16238 | 28851 | 18675 | 13366 | 9982 | 270225 |
| $S5$ | 31564 | 20341 | 28416 | 42543 | 58191 | 39386 | 72631 | 69819 | 33780 | 56143 | 38839 | 27847 | 21588 | 541088 |
| $S6$ | 65969 | 45720 | 60557 | 88389 | 118403 | 81016 | 144117 | 145212 | 73501 | 114376 | 81466 | 61508 | 47754 | 1127988 |
| $S7$ | 11628 | 6782 | 9571 | 15949 | 22680 | 13918 | 29550 | 26915 | 11801 | 22604 | 13019 | 8752 | 6403 | 199572 |
| $S8$ | 23245 | 15294 | 21369 | 32093 | 43139 | 29085 | 54114 | 52438 | 25327 | 41967 | 28706 | 20331 | 15708 | 402816 |
| $S9$ | 51459 | 33572 | 45587 | 68692 | 96131 | 64206 | 117659 | 115680 | 56550 | 91688 | 63418 | 45788 | 35737 | 886167 |

Similarly, we can compute the number of the occurrences in all the subnettrees and do not introduce them in detail. From this example, we can see that SETA is a very effective algorithm.

## 5 Experimental results and analysis

### 5.1 Experimental environment and data

The data used in this paper are real biological sequences provided by the National Center for Biotechnology Information website. Homo Sapiens AX829174, AL158070 and AB038490 are chosen as our test data and can be downloaded from http://www.ncbi.nlm.nih.gov/nuccore/AX829174, http://www.ncbi.nlm.nih.gov/nuccore/AL158070.11, http://www.ncbi.nlm.nih.gov/nuccore/AB038490, respectively. To show how the different lengths of sequence affect the running time, we divide each sequence into 3 segments which are about 25 %, 50 %, and 100 % of the original length, respectively. Therefore, there are 9 sequences shown in Table 2. The source codes of SETA can be obtained from http://wuc.scse.hebut.edu.cn/nettree/sap/. All experiments are run on a laptop with Intel(R)Core(TM)2 DuoT6670@2.20GHz CPU and 3.0GB of RAM, Windows 7.

### 5.2 Correctness validation

To validate the correctness of SETA, we choose pattern $P1$ =A[-1,2]C[-2,3]G[-3,4]T, $Minlen = 4$, $Maxlen$ =10, and threshold $d = 1$ and calculate the number of approximate occurrences in the sequences $S1$~$S9$. According to Theorem 1, this instance can be transformed into the following 13 instances of SPANGLO $P1$, $Q1$=C[-1,2]C[-2,3]G[-3,4]T, $Q2$=G[-1,2]C[-2,3]G[-3,4]T, $Q3$=T[-1,2]C[-2,3]G[-3,4]T, $Q4$=A[-1,2]A[-2,3]G[-3,4]T, $Q5$=A[-1,2]G[-2,3]G[-3,4]T, $Q6$=A[-1,2]T[-2,3]G[-3,4]T, $Q7$=A[-1,2]C[-2,3]A[-3,4]T,

**Table 4** Running time of SETS and SETA (ms)

| | SETS | | | | | | | | | | | | | SET A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sequence | $P1$ | $Q1$ | $Q2$ | $Q3$ | $Q4$ | $Q5$ | $Q6$ | $Q7$ | $Q8$ | $Q9$ | $Q10$ | $Q11$ | $Q12$ | $P1, d$=1 |
| $S1$ | 31 | 46 | 16 | 28 | 26 | 34 | 36 | 34 | 34 | 39 | 33 | 36 | 36 | 64 |
| $S2$ | 46 | 46 | 46 | 84 | 84 | 56 | 82 | 55 | 62 | 53 | 54 | 54 | 53 | 142 |
| $S3$ | 78 | 93 | 109 | 146 | 132 | 92 | 132 | 96 | 114 | 112 | 146 | 86 | 172 | 203 |
| $S4$ | 374 | 406 | 421 | 344 | 413 | 456 | 364 | 427 | 364 | 385 | 318 | 347 | 315 | 955 |
| $S5$ | 764 | 718 | 890 | 676 | 926 | 708 | 703 | 838 | 787 | 736 | 662 | 672 | 707 | 2206 |
| $S6$ | 1701 | 1653 | 2324 | 1457 | 1646 | 1420 | 1594 | 1569 | 1379 | 1617 | 1517 | 1397 | 1518 | 3987 |
| $S7$ | 250 | 265 | 230 | 258 | 362 | 272 | 365 | 318 | 237 | 308 | 313 | 278 | 282 | 664 |
| $S8$ | 562 | 578 | 665 | 522 | 526 | 554 | 563 | 557 | 485 | 784 | 627 | 472 | 537 | 1364 |
| $S9$ | 1264 | 1216 | 1201 | 1065 | 1329 | 1175 | 1396 | 1203 | 1127 | 1317 | 53 | 1016 | 1054 | 2646 |

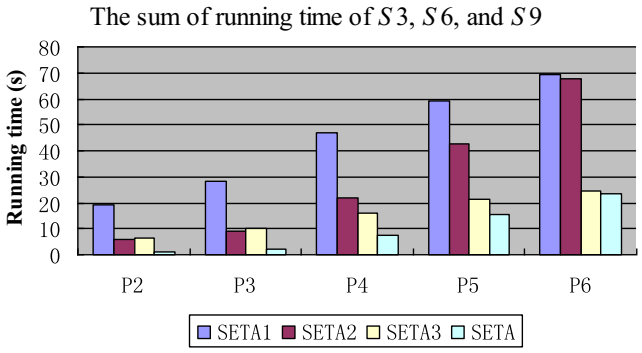



**Fig. 3** Comparison of the running time

$Q8$=A[-1,2]C[-2,3]C[-3,4]T, $Q9$=A[-1,2]C[-2,3]T[-3,4]T, $Q10$=A[-1,2]C[-2,3]G[-3,4]A, $Q11$=A[-1,2]C[-2,3]G[-3,4]C and $Q12$=A[-1,2]C[-2,3]G[-3,4]G. We calculate SPANGLO of the 13 patterns in the 9 sequences using SETS and SAP of pattern $P1$ using SETA. The results are shown in Table 3. In order to validate the effectiveness of SETA, Table 4 shows the running time.

As shown in Table 3, the results validate the correctness of not only Theorem 1 but also SETA. Taking the longest sequence $S6$ as an example, the sum of 13 SPANGLO instances using SETS is 1127988, and of SAP instance using SETA is also 1127988. We have the same results with the other 8 sequences. Therefore, both Theorem 1 and SETA are correct. On the other hand, when the length of pattern, the size of alphabet and threshold are 4, 4, and 1, respectively, a SAP instance can be transformed into $1+4^*(4-1)=13$ SPANGLO instances. Hence, when one of $d$, $m$ or $|\sum|$ increases, the number of the corresponding SPANGLO instances of a SAP instance will increase rapidly, especially when $d$ increases, and the corresponding instances will increase exponentially. Therefore, it is essential to propose an algorithm to deal with SAP.

Table 4 can adequately demonstrate the effectiveness of SETA. Still taking the longest sequence $S6$ as an example, if we adopt SETS to calculate the running times of the 13 patterns, the sum of the running times is 20792(ms), while when we adopt SETA to calculate, the running time is only 3987(ms), which is about 5 times faster than the sum of the running times of SETS. As said above, when one of $d$, $m$ or $|\sum|$ increases, SETA is superior to SETS significantly.

### 5.3 Effectiveness

We know that Lemma 1 and 6 are two pruning strategies. To show how the two strategies affect the running time we also propose three algorithms named SETA1, SETA2, and SETA3. Neither Lemma 1 nor Lemma 6 is employed in SETA1, while they are employed in SETA2 and SETA3, respectively. We select 5 patterns with different numbers of negative gaps which are $P2$ = G[0, 2]A[0, 1]C[1, 2]G[0, 2]T[0, 1]C[0, 1]C[0, 2]A[0, 1]C, $P3$ = G[−2, 2]A[−2, 1]C[1, 2]G[0, 2]T[0, 1]C[0, 1]C[0, 2]A[0, 1]C, $P4$ = [−2, 2]A[−2, 1]C[−2, 2]G[−2, 2]T[0, 1]C[0, 1]C[0, 2]A[0, 1]C, $P5$ = G[−2, 2]A[−2, 1]C[−2, 2]G[−2, 2]T[−2, 1]C[−2, 1]C[0, 2]A[0, 1]C and $P6$ = G[−2, 2]A[−2, 1]C[−2, 2]G[−2, 2]T[−2, 1]C[−2, 1]C[−2, 2] A[−2, 1]C. The length constraint and threshold are $Minlen = 11$, $Maxlen = 16$, and $d = 1$, respectively. Figure 3 shows the sum of the running times of these patterns on $S3$, $S6$, and $S9$.

We know that $P2$, $P3$, $P4$, $P5$, and $P6$ have 0, 2, 4, 6, and 8 negative gaps, respectively. From Fig. 3, we know that the running time tends to increase along with the number of negative gaps. Since Lemma 1 prunes the subnettree according to the pattern, the less the number of negative gaps is the more effective the strategy is. Therefore, we can see that SETA2 is more effective on $P2$, but less effective on $P6$. The most important conclusion is that SETA is the most effective algorithm. The reason is that SETA employs two effective pruning strategies.

**Table 5** Running time for different lengths of patterns(ms)

| Sequence | $m=4$ | $m=5$ | $m=6$ | $m=7$ | $m=8$ | $m=9$ | $m=10$ | $m=11$ |
|---|---|---|---|---|---|---|---|---|
| $S1$ | 114 | 127 | 232 | 234 | 346 | 647 | 648 | 839 |
| $S2$ | 133 | 274 | 335 | 587 | 699 | 953 | 1285 | 1692 |
| $S3$ | 252 | 675 | 784 | 1141 | 1542 | 1893 | 2954 | 3647 |
| $S4$ | 1053 | 1764 | 3155 | 4326 | 5687 | 7647 | 10769 | 14350 |
| $S5$ | 1696 | 3644 | 5909 | 8427 | 11613 | 15321 | 22321 | 28112 |
| $S6$ | 3283 | 6884 | 11796 | 16577 | 24599 | 31404 | 45773 | 55217 |
| $S7$ | 612 | 1343 | 2205 | 3468 | 4376 | 5685 | 9164 | 10053 |
| $S8$ | 1454 | 2325 | 4405 | 6776 | 9067 | 11617 | 16163 | 20252 |
| $S9$ | 2766 | 5541 | 9361 | 14062 | 18392 | 24906 | 37117 | 46365 |

**Table 6** Results for different lengths of patterns

| Sequence | $m = 4$ | $m = 5$ | $m = 6$ | $m = 7$ | $m = 8$ | $m = 9$ | $m = 10$ | $m = 11$ |
|---|---|---|---|---|---|---|---|---|
| $S1$ | 13890 | 19590 | 30401 | 41570 | 59717 | 95448 | 130785 | 194842 |
| $S2$ | 28553 | 39430 | 63455 | 89074 | 127091 | 206747 | 287104 | 436800 |
| $S3$ | 56201 | 88956 | 143164 | 180591 | 277684 | 459567 | 577332 | 907282 |
| $S4$ | 231248 | 425279 | 682651 | 850517 | 1527276 | 2451005 | 3046075 | 5482153 |
| $S5$ | 469373 | 837077 | 1342584 | 1721961 | 2987121 | 4803078 | 6167334 | 10729631 |
| $S6$ | 966145 | 1692465 | 2720641 | 3500327 | 5949884 | 9678170 | 12464445 | 21032527 |
| $S7$ | 170123 | 326085 | 521720 | 628157 | 1179502 | 1876797 | 2238738 | 4238587 |
| $S8$ | 346278 | 624062 | 989769 | 1248636 | 2193298 | 3486644 | 4388263 | 7736713 |
| $S9$ | 766400 | 1359543 | 2202630 | 2802655 | 4818598 | 7879130 | 10034750 | 17130517 |

## 5.4 Evaluation

In this section, generally, we neglect *Minlen* and *Maxlen*; besides, the threshold is $d = 1$.

### *5.4.1 Lengths of pattern and sequence evaluation*

To show how the length of pattern affects the results and running time, we choose pattern $P7$=G[-1,3]T[-1, 3]A[-1,3]G[-1,3]T[-1,3]A[-1,3]G[-1,3]T[-1,3]A[-1,3]G[-1,3]T whose length is 11 and its sub-patterns; for instance, if $m$=4, it indicates that the length of the prefix pattern of $P7$ is 4, i.e. G[-1,3]T[-1,3]A[-1,3]G. We show the running time and results in Tables 5 and 6, respectively.

From Table 5, we can clearly see that the running time of SETA is in linear growth with the length of sequence. For example, when $m = 4$, the running times on sequences $S7$, $S8$, and $S9$ are 612, 1454, and 2766 ms, respectively, which are in linear growth with the length of sequence. We also notice that the running time grows quadratically with $m$. Hence, these experimental results validate the correctness of the time complexity of SETA.

We can see from Table 6 that, as the length of pattern increases, the solution of SAP increases rapidly. Especially, we notice an interesting phenomenon that when $m$ increases by 2 every time, the solution of SAP will enlarge to about twice the previous. Taking sequence $S1$ as an example, when $m = 4$, the result is 13890, while when $m = 6$, the result increases to 30401 which is 2.2 times the previous value, and most instances in this table have the same phenomenon. Besides, a more outstanding phenomenon is that the solution of SAP is in linear growth with the length of sequence. We can see from Table 2 that the length of $S2$ is twice that of $S1$, while the result of SAP on $S2$ is also about twice that on $S1$. All other instances in Table 6 also present this phenomenon. Hence the solution of SAP is about $n*W\hat{}(m-1)$.

### *5.4.2 Threshold evaluation*

In order to show how the threshold affects the running time and results, we use pattern $P8$ = G[−1, 3]T[−1, 3]A[−1, 3]G[−1, 3]T[−1, 3]A[−1, 3]G[−1, 3]T[−1, 3]A[−1, 3]G[−1, 3]T, *Minlen*= 11, *Maxlen* = 16, and $d = 0, d = 1, d = 2, d = 3, d = 4$, and $d = 5$, respectively. The running time and results are shown in Tables 7 and 8, respectively.

We can see from Table 7 that the running time of SETA is in linear growth with $d$. Taking $S3$ as an example, when $d = 2$, the running time is 3948 ms, which is about 2 times that of $d = 1$. Similarly, the running times of $d = 3$, $d = 4$, and $d = 5$ are about 3 times, 4 times and 5 times that of $d = 1$. Besides, we notice that the running times from $d = 0$ to $d = 1$ change significantly. The reason is that when $d = 0$, this is exact matching and SETA conducts pruning according to Lemma 6, which can improve the speed. In summary, the experiments validate that the running time is in linear growth with $d$.

**Table 7** Running time for different thresholds (ms)

| Sequence | $d = 0$ | $d = 1$ | $d = 2$ | $d = 3$ | $d = 4$ | $d = 5$ |
|---|---|---|---|---|---|---|
| $S3$ | 537 | 1943 | 3984 | 5761 | 7542 | 8596 |
| $S6$ | 7829 | 32202 | 65725 | 95489 | 120745 | 144255 |
| $S9$ | 7683 | 25341 | 52796 | 83851 | 95832 | 114505 |

**Table 8** Results for different thresholds

| Sequence | $d = 0$ | $d = 1$ | $d = 2$ | $d = 3$ | $d = 4$ | $d = 5$ |
|---|---|---|---|---|---|---|
| $S3$ | 12304 | 280355 | 3493158 | 30280999 | 191689519 | 889831210 |
| $S6$ | 255555 | 6129133 | 74497830 | 614483222 | 3699685835 | 16439213944 |
| $S9$ | 202185 | 4907814 | 60097318 | 498145227 | 3003339685 | 13321967209 |

From Table 8, we can see that the result of SAP increases rapidly with $d$. Taking $S3$ as an example, when $d$ varies from 2 to 3, the result increases about 8.7 times. The reason is that a SAP instance can be transformed into $C_m^d$ SPANGLO instances. Hence, the result of SAP is in exponential growth with $d$.

## 6 Conclusion

In this paper, we propose the SAP problem which is a strict approximate pattern matching with general gaps and length constraints. We prove that a SAP instance can be converted to exponential exact matching instances and design an effective online algorithm, named SETA, which employs the subnettree structure and adopts many efficient pruning strategies to deal with SAP online. We analyse the time and space complexities of SETA, which are $O(Maxlen \times W \times m^2 \times n \times d)$ and $O(m \times Maxlen \times W \times d)$, respectively, where $m$, $Maxlen$, $W$, and $d$ are the length of pattern $P$, the maximal length constraint, the maximal gap length of pattern $P$ and the approximate threshold, respectively. Besides, extensive experimental results validate the correctness and completeness of SETA, and the contrast experiments validate the effectiveness of SETA. Finally, we also illustrate how $m$, $n$, and $d$ affect the results and running time.

In the future, we will focus on mining approximate sequential patterns with general gaps, especially for larger sequences [29]. Besides, while this paper focuses on strict pattern matching without special condition, there are also types of strict pattern matching with the non-overlapping condition or the one-off condition which are worth exploring.

**Acknowledgments** This research is supported by the National Natural Foundation of China under grants No. 61229301 and 61370144, the Program for Changjiang Scholars and Innovative Research Team in University (PCSIRT) of the Ministry of Education, China, under grant IRT13059, the Natural Science Foundation of Hebei Province of China under grant No. F2013202138, and the Key Project of the Educational Commission of Hebei Province under grant No. ZH2012038.

## References

1. Chouvalit K, Veera B (2013) A new linear-time dynamic dictionary matching algorithm. Comput Inform 32(5):897–923
2. Aligon J, Golfarelli M, Marcel P, Rizzi S, Turricchia E (2014) Similarity measures for OLAP sessions. Knowl Inf Syst 39(2):463–489
3. Knuth DE, Morris JH, Pratt VR (1977) Fast pattern matching in strings. SIAM J. Comput 6(2):323–350
4. Fischer MJ, Paterson MS (1974) String matching and other products. In: Proceedings of the 7th SIAM AMS complexity of computation, Cambridge, USA, pp 113–125
5. Manber U, Baeza YR (1991) An algorithm for string matching with a sequence of don't cares. Inf Process Lett 37(2):133–136
6. Navarro G, Raffinot M (2003) Fast and simple character classes and bounded gaps pattern matching with applications to protein searching. J Comput Biol 10(6):903–923
7. Cole R, Gottlieb L, Lewenstein M (2004) Dictionary matching and indexing with errors and don't cares. In: Proceedings of the 36th ACM symposium on the theory of computing, Chicago, USA, pp 91–100
8. Crochemore M, Iliopoulos C, Makris C, Rytter W, Tsakalidis A, Trichlas K (2002) Approximate string matching with gaps. Nord J Comput 9(1):54-65
9. Cantone D, Cristofaro S, Faro S (2009) New efficient bit-parallel algorithms for the $(\delta, \alpha)$-matching problem with applications in music information retrieval. Int J Found Comput Sci 20(6): 1087–1108
10. Ji X, Bailey J, Dong G (2007) Mining minimal distinguishing subsequence patterns with gap constraints. Knowl Inf Syst 11(2):259–286
11. Ferreira PG, Azevedo PJ (2005) Protein sequence pattern mining with constraints. In: European conference on principles and practice of knowledge discovery in databases (PKDD), Porto, Portugal, pp 96–107
12. Zhang M, Kao B, Cheung D, Yip K (2007) Mining periodic patterns with gap requirement from sequences. ACM Trans Knowl Discov Data 1(2):7–es
13. Zhu X, Wu X (2007) Mining complex patterns across sequences with gap requirements. In: Proceedings of the 20th international joint conference on artificial intelligence (IJCAI), Hyderabad, India, pp 2934–2940
14. Wu Y, Wang L, Ren J, Ding W, Wu X (2014) Mining sequential patterns with periodic wildcard gaps. Appl Intell 41(1):99–116
15. Tsai CY, Chen CJ, Chien CJ (2013) A time-interval sequence classification method. Knowl Inf Syst 37(2):251–278
16. Wu Y, Liu Y, Guo L, Wu X (2013) Subnettrees for strict pattern matching with general gaps and length constraints. J Softw 24(5):915–932
17. Fredriksson K, Grabowski S (2006) Efficient algorithms for pattern matching with general gaps and character classes. In: International conference on string processing and information retrieval, Glasgow, UK, pp 267–278

18. Fredriksson K, Grabowski S (2008) Efficient algorithms for pattern matching with general gaps, character classes, and transposition invariance. Inf Retrieval 11(4):335–357
19. Guo D, Hu X, Xie F, Wu X (2013) Pattern matching with wildcards and gap-length constraints based on a centrality-degree graph. Appl Intelligence 39(1):57–74
20. Huang Y, Wu X, Hu X, Xie F, Gao J, Wu G (2009) Mining frequent patterns with gaps and one-off condition. In: IEEE international conference on computational science and engineering (CSE'09), Vancouver, BC, Canada, pp 180–186
21. Lam HT, Mörchen F, Fradkin D (2014) Mining compressing sequential patterns. Stat Anal Data Min 7(1):34–52
22. Ding B, Lo D, Han J (2009) Efficient mining of closed repetitive gapped subsequences from a sequence database. In: IEEE 25th international conference on data engineering (ICDE), Shanghai, China, pp 1024–1035
23. Min F, Wu X, Lu Z (2009) Pattern matching with independent wildcard gaps. In: Proceedings of the 8th international conference on pervasive intelligence and computing, Chengdu, China, pp 194–199
24. Bille P, Gørtz I, Vildhøj H, Wind D (2010) String matching with variable length gaps. In: Proceedings of the 17th international conference on string processing and information retrieval, SPIRE, Mexico, pp 385–394
25. Rahman S, Iliopoulos C, Lee I, Mohamed M, Smyth W (2006) Finding patterns with variable length gaps or don't cares. In: 12th annual international conference computing and combinatorics, Taiwan, pp 146–155
26. Bille P, IL Gørtz, Vildhøj HW (2012) String matching with variable length gaps. Theor Comput Sci 443:25–34
27. He D, Wu X, Zhu X (2007) SAIL-APPROX: An efficient online algorithm for approximate pattern matching with wildcards and length constraints. In: Proceedings of the 2007 IEEE international conference on bioinformatics and biomedicine (BIBM'07), Silicon Valley, USA, pp 151–158
28. Wu Y, Wu X, Min F, Li Y (2010) A Nettree for pattern matching with flexible wildcard constraints. In: Proceedings of the 2010 IEEE international conference on information reuse and integration (IRI2010), Las Vegas, USA, pp 109–114
29. Rasheed F, Adnan M, Alhajj R (2013) Out-of-core detection of periodicity from sequence databases. Knowl Inf Syst 36(1): 277–301

**Youxi Wu**, born in 1974, Ph. D., is a Professor of Computer Science at Hebei University of Technology. His research interests include data mining and intelligent computation.

**Shuai Fu**, born in 1988, is a Masters degree candidate at Hebei University of Technology. His research interests include data mining.

**He Jiang**, born 1980, Ph.D., is a Ph. D. supervisor and a Professor of Computer Science at Dalian University of Technology. His research interests include intelligent computation and software engineering.

**Xindong Wu**, is a Yangtze River Scholar in the School of Computer Science and Information Engineering at the Hefei University of Technology (China), a Professor of Computer Science at the University of Vermont (USA), and a Fellow of the IEEE and the AAAS. He holds a PhD in Artificial Intelligence from the University of Edinburgh, Britain. His research interests include data mining, Big Data analytics, knowledge-based systems, and Web information exploration.