Article

JIS

Journal of Information Science
2016, Vol. 42(5) 639–658

DOI: 10.1177/0165551515603286
jis.sagepub.com
SAGE

Approximate pattern matching with gap constraints

Youxi Wu
Hebei University of Technology, China

Zhiqiang Tang
Hebei University of Technology, China

He Jiang
Dalian University of Technology, China

Xindong Wu
Hefei University of Technology, China
University of Vermont, Vermont, USA

Abstract

Pattern matching is a key issue in sequential pattern mining. Many researchers now focus on pattern matching with gap constraints. However, most of these studies involve exact pattern matching problems, a special case of approximate pattern matching and a more challenging task. In this study, we introduce an approximate pattern matching problem with Hamming distance. Its objective is to compute the number of approximate occurrences of pattern P with gap constraints in sequence S under similarity constraint d. We propose an efficient algorithm named Single-rOot Nettree for approximate pattern matchinG with gap constraints (SONG) based on a new non-linear data structure Single-root Nettree to effectively solve the problem. Theoretical analysis and experiments demonstrate an interesting law that the ratio $M(P,S,d)/N(P,S,m)$ approximately follows a binomial distribution, where $M(P,S,d)$ and $N(P,S,m)$ are the numbers of the approximate occurrences whose distances to pattern P are d ($0 \leq d \leq m$) and no more than m (the length of pattern P), respectively. Experimental results for real biological data validate the efficiency and effectiveness of SONG.

Keywords

Approximate pattern matching; Gap constraints; Length constraint; Hamming distance; Nettree

1. Introduction

With the development of information technology, the amount of information which must be processed is increasing rapidly. For instance, in the field of biology there are vast quantities of DNA and RNA sequences to analyse, including those of various viruses (SARS, H1N1, etc.) that threaten our lives and genetic information. Both pattern matching and pattern mining techniques can extract useful information from massive data [1] and pattern matching is a key issue in sequential pattern mining [2]. Pattern matching is one of the essential problems in computer science with broad applications, such as bug detection [3], ontology matching [4], etc. Therefore, many researchers focus on improving the performance of pattern matching [5]. When wildcards (also known as 'don't care' characters) are introduced into the pattern, it becomes more practical, since these wildcards can give users more flexibility to control their queries. Therefore, pattern matching with wildcards significantly impacts on many applications, including biological sequence analysis [6], text indexing [7], sequential pattern mining [2] and information retrieval [8].

Corresponding author:
Youxi Wu, School of Computer Science and Engineering, Hebei University of Technology, Tianjin 300130, China.
Emails: wuc@scse.hebut.edu.cn; wuc567@163.com

Although '?' and '*' are two common wildcards, it is difficult to use them to present gap constraints (or flexible wildcards). Therefore, *min* and *max* are employed to express the minimal and maximal numbers of wildcards, respectively, between two consecutive characters. A pattern with gap constraints used in pattern matching and sequential pattern mining can be expressed as $p_0[a_0, b_0]p_1\ldots[a_{j-1}, b_{j-1}]p_j\ldots[a_{m-2}, b_{m-2}]p_{m-1}$, where a_j and b_j are a group of gap constraints. In some pattern matching studies, such as [9–11], the position of the last pattern substring in the sequence is considered to represent an occurrence. This type of study is called loose pattern matching [12]. Another type of study is strict pattern matching, in which a group of position indices of the sequence is used to represent an occurrence. Min et al. [13], Wu et al. [14], and Guo et al. [15] employ such strict pattern matching techniques. Apparently, whereas loose pattern matching ignores the matching process in detail, strict pattern matching does not. It is easy to see that the number of occurrences is no more than the length of sequence S in a loose pattern matching problem. However, the number of occurrences can be exponential with regard to pattern length in a strict pattern matching problem. Hence loose pattern matching is far simpler than strict pattern matching.

Strict pattern matching plays an essential role in many critical sequential pattern mining tasks. For instance, Zhang et al. [16] and Ji et al. [17] proposed methods for mining periodic patterns with gap constraints. Li et al. [18] proposed two algorithms to mine closed and repetitive gap-constrained sub-sequences which can be used in feature selection for the purpose of classification and clustering. Zhang et al. [16] addressed the mining of frequent patterns with periodic wildcard gaps. The most important task in solving this problem involves calculating the number of occurrences in order to determine whether the pattern is frequent or not. Zhu and Wu [19] proposed the GCS algorithm in which a pattern matching method was used to calculate the number of occurrences and to solve the issue outlined in [16] more effectively. Wu et al. [2] proposed an algorithm, named MAPD, which also employs a pattern matching strategy and is more effective than other competitive algorithms. Pattern matching with gap constraints can therefore be considered to represent the foundation of sequential pattern mining with gap constraints.

Generally, pattern matching can be either exact or approximate. Exact pattern matching is essentially a special case of approximate pattern matching, since if the similarity constraint is 0, an approximate pattern matching instance will be an exact pattern matching instance. Approximate pattern matching is more complex yet more meaningful than exact pattern matching. Many things in the real world have the same functions with similar structures. For instance, the DNA or RNA sequences of viruses may mutate constantly without changing any of their basic features. We know that exact pattern matching can hardly be used to solve approximate pattern matching problems. Hence, we are interested in a new approximate pattern matching technique with gap constraints. The following part presents an example to illustrate our study.

Example 1. Given pattern $P=p_0[a_0,b_0]p_1[a_1,b_1]p_2$=a[0,2]g[1,3]a, sequence $S=s_0s_1s_2s_3s_4s_5s_6$=atggaga, length constraints *MinLen*=4 and *MaxLen*=8, and similarity constraint d=1.

First, if we do not consider the similarity constraint or the similarity constraint d=0, the instance is an exact pattern matching problem which was solved in [13, 14]. According to pattern P and sequence S, we know that pattern P has two gaps. Each gap has two gap constraints, one being the lower bound of the number of wildcards and the other the upper bound of the number of wildcards. For example, for the first group of gap constraints, [0,2] in 'a[0,2]g' means that the number of wildcards between 'a' and 'g' can be 0, 1 or 2. We use a group of position indices in sequence S to denote an occurrence. The span (the distance between the beginning position and ending position) of an occurrence should be subject to the length constraints. For instance, $<0,2,4>$ is an exact occurrence of P in S, because $p_0=s_0$=a, $p_1=s_2$=g, $p_2=s_4$=a, $a_0 \le 2\text{-}0\text{-}1 \le b_0$ and $a_1 \le 4\text{-}2\text{-}1 \le b_1$. $<0,1,4>$ is not an exact occurrence, although it does satisfy the gap constraints because p_1='g' is not equal to s_1='t'. However, we can say that $<0,1,4>$ is an approximate occurrence with a Hamming distance of 1. Similarly, $<0,4,6>$ is not an occurrence, because 4-0-1=3 is not subject to gap constraints $[a_0, b_0]$. We can thus easily enumerate all three exact occurrences: $<0,2,4>$, $<0,2,6>$ and $<0,3,6>$, as well as all seven approximate occurrences with a Hamming distance of 1: $<0,1,4>$, $<0,2,5>$, $<0,3,5>$, $<1,2,4>$, $<1,2,6>$, $<1,3,6>$ and $<2,3,6>$. So there are 10 occurrences under the similarity constraint d=1. In the loose pattern matching problem, all occurrences are 4, 5 and 6. The span of occurrence $<0,2,4>$ is 4-0 + 1=5, which is subject to *MinLen*=4 and *MaxLen*=8, since $MinLen \le 5 \le MaxLen$. Suppose the length constraints are *MinLen*=4 and *MaxLen*=6; occurrences $<0,2,6>$ and $<0,3,6>$ are not subject to these constraints since the spans of occurrences $<0,2,6>$ and $<0,3,6>$ are both 7, which is greater than *MaxLen*=6. Therefore, there are eight occurrences under *MinLen*=4 and *MaxLen*=6.

From this example, we learn that the problem has the following four characteristics:

1. The pattern has multiple gaps. For instance, in Example 1, pattern P has two gaps: [0,2] and [1,3].
2. Each occurrence should be subject to the length constraints. Although all gap constraints can determine the length constraints, users cannot control their occurrence spans effectively. The present study therefore considers the length constraints.

3. Any position index for sequence S can be used more than once. For instance, index 0 is used in occurrences $<0,1,4>$, $<0,2,4>$, $<0,2,5>$, $<0,2,6>$, $<0,3,5>$ and $<0,3,6>$, i.e. six times. However, in [15, 20], each index can be used at most once; this constraint is known as the one-off condition.
4. The most important characteristic of the problem is that it is essentially a type of strict approximate pattern matching with Hamming distance.

In summary, the contributions of this paper are as follows:

First, we issue a new problem of approximate pattern matching with gap constraints. As well as gap constraints between every two consecutive characters in the pattern, the characters in the pattern can also be matched approximately, in terms of the Hamming distance. We also consider the length constraints to provide users with more flexibility. Since the size of all the approximate occurrences is exponential with regard to pattern length, our goal is to compute their number.

Second, we analyse the ratio $M(P,S,d)/N(P,S,m)$ theoretically, where $M(P,S,d)$ and $N(P,S,m)$ are the numbers of the approximate occurrences whose distances to pattern P are d ($0 \leq d \leq$ m) and no more than m, respectively. It follows the binomial distribution $B(m, 1\text{-}q)$, where $q=\sum_{l=1}^{\lambda} U(c_l) \times V(c_l)$, $U(c_l)$ and $V(c_l)$ are the probabilities of character c_l in pattern P and sequence S, respectively (which means that the probability of getting exactly d successes in n trials is $C_m^d \times (1-q)^d \times q^{m-d}$, where $C_m^d=\frac{m!}{d!(m-d)!}$). Experimental results obtained via the use of real biological data also verify the correctness of this analysis.

Finally, to solve this problem, we propose a new algorithm named Single-rOot Nettree for approximate pattern matchinG with gap constraints (SONG), which transforms the problem into many Single-root Nettrees, using the new concepts of the latter to solve the problem. The time and space complexities of SONG are $O((n\text{-}m)*m^2*W^2*d)$ and $O(m^2*W*d)$, respectively, where n, m, W and d are sequence length, pattern length, the maximal gap of pattern P and the similarity constraint, respectively. Experimental results on real biological data validate the efficiency and correctness of SONG.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the problem statement with an illustrative example for the definitions and theoretically analyses the problem. Section 4 defines the Single-root Nettree and introduces new concepts. On this basis, we then propose the algorithm of SONG and analyse its time and space complexities. Moreover, we demonstrate how it works via a running example. Section 5 presents the experimental results and analysis over real biological data. Conclusions are drawn in Section 6.

2. Related work

Considering that pattern matching is a key issue in sequential pattern mining, Table 1 compares related studies focusing on pattern matching and sequential pattern mining with gap constraints.

Fischer and Paterson [21] were the first to use wildcards in pattern matching. However, in their case the number of wildcards between two consecutive characters was kept constant. Manber et al. [22] replaced this constant with a flexible

Table 1. Comparison of related algorithms.

Algorithms	Matching / Mining	Number of gaps	Gap constraint	Length constraints	Type of matching	Type of condition
Fischer et al. [21]	Matching	Single	No	No	Strict exact matching	–[a]
Manber et al. [22]	Matching	Single	Yes	No	Strict exact matching	–
Bille et al. [23]	Matching	Multiple	Yes	No	Loose / Strict exact matching[b]	–/ No condition
Guo et al. [15]	Matching	Multiple	Yes	Yes	Strict exact matching	One-off
Ding et al. [24]	Mining	Multiple	Yes	Yes	Strict exact matching	Non-overlapping
He et al. [20]	Matching	Multiple	Yes	Yes	Strict approximate matching with Hamming distance	One-off
Zhang et al. [16]	Mining	Multiple	Yes	Yes	Strict exact matching	No condition
Wu et al. [14]	Matching	Multiple	Yes	Yes	Strict exact matching	No condition
This paper	Matching	Multiple	Yes	Yes	Strict approximate matching with Hamming distance	No condition

[a]'–' in the table represents the items we do not take into consideration.

[b]This paper designed two algorithms, which research loose pattern matching and strict pattern matching, respectively.

wildcard gap, although in this case the pattern can have only one gap. On the basis of [21, 22], pattern matching with multiple variable-length gaps has subsequently attracted much research attention, including [9, 10, 13, 20, 23]. Studies [9, 10] were able to find the occurrences of a pattern in a sequence. Since the number of occurrences of a pattern with multiple groups of gap constraints is exponential, the traditional method of finding all the occurrences is unfeasible. As a result, [9, 10] compacted the size of occurrences by locating only their end positions, with each position located only once. Hence, the latter two studies belong to the loose exact matching type.

Computing the exponential number of all occurrences of a pattern with independent and gap constraints is a type of strict pattern matching. Min et al. [13] and Wu et al. [14] separately designed two algorithms able to effectively solve the exact matching problem. Their methods employed to solve the issue outlined in [16]. However, their methods could not handle the approximate matching problem. The problem presented here can turn into theirs if the similarity constraint d is equal to 0. The present paper therefore deals with a more general strict pattern matching problem with Hamming distance.

In strict pattern matching there are three types of condition: the one-off condition, the non-overlapping condition and no condition. The one-off condition means that any position in the sequence can be used at most once; the total number of occurrences under this condition is thus no greater than n/m. The one-off condition has many different names in sequential pattern mining, with Ferreira and Azevedo [25], Wu et al. [26] and Lam et al. [27] all researching pattern mining with the one-off condition. The non-overlapping condition [24] means that any position in the sequence can be used more than once, but cannot be reused by the same sub-pattern. Hence, the total number of occurrences under this condition is no greater than n. No condition means that any position can be used more than once without any condition. From Table 1, it can be observed that whereas He et al. [20] considered the one-off condition, we focus on no condition. The problem examined in the present paper can thus be seen as computing the candidate space of [20].

3. Problem

In this section, we first give a brief definition of approximate pattern matching with gap constraints. We then theoretically analyse the problem.

3.1. Problem definition

Definition 1. A pattern can be denoted as $P=p_0[a_0, b_0]p_1\ldots [a_{j\text{-}1}, b_{j\text{-}1}]p_j\ldots [a_{m\text{-}2}, b_{m\text{-}2}]p_{m\text{-}1}$, where m denotes the length of P, $a_{j\text{-}1}$ and $b_{j\text{-}1}$ are given integers in a group of gap constraints, representing the minimal and maximal wildcards between p_j and p_{j+1}, where $0 \le a_j \le b_j$.

Definition 2. A sequence can be denoted as $S=s_0s_1\ldots s_i\ldots s_{n\text{-}1}$, where n is the length of S. $\sum$ represents a set of characters and λ is the length of $\sum$.

Definition 3. Given two sequences $P=p_0p_1\ldots p_{m\text{-}1}$ and $Q=q_0q_1\ldots q_{m\text{-}1}$, if there are k positions at which the corresponding characters are different, *i.e.* $p_i \neq q_i$ $(0 \le i < m)$, then the Hamming distance between the two strings is k $(0 \le k \le m)$. $D(P,Q)$ is used to denote the Hamming distance between P and Q.

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

$$D(p_0\cdots p_j\cdots p_{m-1}, S_{i_0}\cdots S_{i_j}\cdots S_{i_{m-1}}) \le d \tag{1}$$

$$0 \le a_{j-1} \le i_j - i_{j-1} - 1 \le b_{j-1} \tag{2}$$

$$0 < MinLen \le i_{m-1} - i_0 + 1 \le MaxLen \tag{3}$$

where $0 \le j \le m\text{-}1$ and $0 \le i_j \le n\text{-}1$, then I is an approximate occurrence of P in S.

Apparently, all gap constraints can determine the length constraints, with the lower bound of *MinLen* and the upper bound of *MaxLen* equal to $m + \sum_{k=0}^{m-2} a_k$ and $m + \sum_{k=0}^{m-2} b_k$, respectively.

Definition 5. The goal of the problem is to compute the total number of approximate occurrences, as denoted by $N(P,S,d)$.

In order to consider the similarity constraint, we define the concepts of similar number and similar branch number.

Definition 6. A sub-occurrence is the group of prefix position indices of an occurrence. Assuming the elements in set E_s are all sub-occurrences from r to i whose length is $j+1$ and the distance between each element and sub-pattern

$p_0p_1\ldots p_j$ is no greater than t, the cardinality of E_s is called the similar number and is denoted by $N_s(r,i,j)$. We define $N_s(r,r,1)=1$. Assuming the elements in set E_b are all sub-occurrences from r to i whose length is $j+1$ and the distance between each element and sub-pattern $p_0p_1\ldots p_j$ is t ($0 \le t \le d$), the cardinality of E_b is called the similar branch number and is denoted by $N_b(r,i,j,d)$. We define $N_b(r,r,1,0)=\begin{cases} 1 & \text{if } p_0 = s_r \\ 0 & \text{else} \end{cases}$, $N_b(r,r,1,1)=\begin{cases} 1 & \text{if } p_0 \neq s_r \\ 0 & \text{else} \end{cases}$ and $N_b(r,r,1,t)=0$($2 \le t \le d$). Apparently, $N_s(r,i,j)= \sum_{t=0}^{d} N_b(r,i,j,t)$.

Example 2. Given P=a[0,2]g[1,3]a, S=atggaga, d=1, *MinLen*=4 and *MaxLen*=8.

We know that $<0,1,4>$ is an approximate occurrence of P in S. However, $<0,1,3>$ is not an approximate occurrence, since $p_1 \neq s_1$ and $p_2 \neq s_3$; therefore, $D(p_0p_1p_2, s_0s_1s_3)=2$. $<0>$ and $<0,1>$ are two sub-occurrences of $<0,1,4>$. According to Definition 6, $N_b(0,4,3,0)=1$ and $N_b(0,4,3,1)=1$; since there are two occurrences, $<0,1,4>$ and $<0,2,4>$, which begin with 0 and end with 4, the distances between them and P are 1 and 0, respectively. Therefore, $N_s(0,4,3)$ equals to 2.

3.2. Theoretical analysis

Let $M(P,S,d)$ denote the number of approximate occurrences whose distances to pattern P are d ($0 \le d \le m$). We know that $N(P,S,d) = M(P,S,0) + M(P,S,1) + \ldots + M(P,S,d)$. It is easy to see that there are $C_m^d \times (\lambda-1)^d$ patterns whose distances to P are d. As a result, the calculation of $M(P,S,d)$ can be transformed into the $C_m^d \times (\lambda-1)^d$ exact pattern matching problem. Assuming that the probability of exact matching for each pattern character in sequence S is $1/\lambda$, we obtain $M(P,S,d)/N(P,S,m)= C_m^d \times [(\lambda-1)/\lambda]^d \times (1/\lambda)^{m-d}$. In this case, $M(P,S,d)/N(P,S,m)$ obviously follows the binomial distribution with m and $(\lambda\text{-}1)/\lambda$, *i.e.* $M(P,S,d)/N(P,S,m) \sim B(m, (\lambda\text{-}1)/\lambda)$. However, in reality the probabilities of different characters in sequence S are usually not equal. Let $V(c_1)$, $V(c_2)$,…, $V(c_\lambda)$ denote the probabilities of λ different characters $c_1,c_2,\ldots,c_\lambda$ in sequence S. Generally, the characters lie randomly in real-life sequences (e.g. DNA, RNA, etc.). Therefore, the probability of successful matching for a certain character c_l ($1 \le l \le \lambda$) of pattern P can be approximated as $V(c_l)$. Let $U(c_l)$ denote the probability of character c_l in pattern P. When we randomly select a character from pattern P and then match it in sequence S, the probability of selecting character c_l and successful matching is approximate to $U(c_l) \times V(c_l)$. The average probability of successful matching a character of pattern P in sequence S is thus approximate to $\sum_{l=1}^{\lambda} U(c_l) \times V(c_l)$ which is denoted as q. Therefore, $M(P,S,d)/N(P,S,m)$ approximately follows the binomial distribution with m and 1-q, i.e. $M(P,S,d)/N(P,S,m) \sim B(m,1\text{-}q)$.

It is easy to see that $M(P,S,0)$ is an exact pattern matching instance which can be solved using PAIG [13]. According to this law, if we know $M(P,S,0)$, then $M(P,S,d)$ and $N(P,S,d)$ ($0 < d \le m$) can be estimated easily. An example will be shown in Section 5.2.3. In order to calculate $M(P,S,d)$ and $N(P,S,d)$ accurately, an effective algorithm is proposed in the following section.

4. Single-root Nettree and SONG

To solve the problem, we must deal with the following three constraints: length constraints, gap constraints and the similarity constraint. As the algorithm would be very complex if all three constraint types were to be tackled simultaneously, the employed strategy is as follows: (1) To handle the length constraints, the beginning position b in sequence S is used to calculate the range of the ending position e according to the length constraints. Therefore, each subsequence from b to e satisfies the length constraints. (2) As Nettree [14] can be used to solve the gap constraints effectively, one was employed here to solve the problem in the sub-sequences. A Nettree with only a root is called a Single-root Nettree, because these sub-sequences have a common beginning position b. Some special concepts are used to control the nodes of a Single-root Nettree. (3) At last, some special lemmas are used to calculate the similar number and similar branch number to handle the similarity constraint in a Single-root Nettree.

4.1. Nettree and Single-root Nettree

Below we first give the formal definition of a Nettree, followed by the definition of a Single-root Nettree. A number of new concepts are also introduced.

Definition 7. A Nettree [14] is an extension of a tree, sharing similar concepts such as root, leaf, level, parent and child. However, a tree differs from a Nettree as follows:

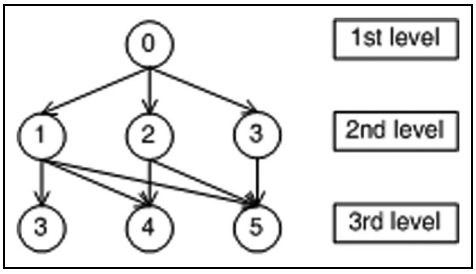

Figure 1. A Single-root Nettree.

1. A Nettree may have more than one root.
2. Some nodes (except roots) in a Nettree may have more than one parent.
3. There may be more than one path from a node to a root.
4. The same node label can occur more than once. Node i on the j-th level is denoted by n_j^i.

Definition 8. A Single-root Nettree is a Nettree with only one root.

Figure 1 shows an example of a Single-root Nettree. Some node labels appear in different levels. For instance, node label 3 occurs on both the second and third levels, with the nodes denoted by n_2^3 and n_3^3, respectively. Some nodes have more than one parent. For instance, Node n_3^4 has two parents: nodes n_2^1 and n_2^2. The Single-root Nettree has one root: n_1^0. Nodes n_3^3, n_3^4 and n_3^5 are three leaves of the Single-root Nettree. There are two paths from root n_1^0 to node n_3^4: $< n_1^0, n_2^1, n_3^4 >$ and $< n_1^0, n_2^2, n_3^4 >$ (written as $< 0,1,4 >$ and $< 0,2,4 >$ in brief).

4.2. Construction of a Single-root Nettree

In this subsection, definitions and lemmas are proposed with which to construct a Single-root Nettree based on the selected problem.

We assume that the sequence indices vary from 0 to n-1. If we create Single-root Nettrees with roots from small to large, one by one, it is easy to see that 0 is the first Single-root Nettree's root. However, n-1 is not the final Single-root Nettree's root, since many positions at the end of the sequence do not satisfy either the length constraints or the gap constraints. In this case we must therefore determine the root range.

Definition 9. The maximal root that satisfies the length constraints and gap constraints is known as the max root and is denoted by *MaxRt*.

Lemma 1. The max root can be calculated via the following equation:

$$MaxRt = \min(n - MinLen, n - m - \sum_{k=0}^{m-2} a_k) \tag{4}$$

Proof: As we know that the maximal ending position is n-1, the maximal beginning position is therefore n-1-$MinLen + 1$ for the minimal span, according to the length constraints. Similarly, another upper bound of the max root is n-m-$\sum_{k=0}^{m-2} a_k$ according to the gap constraints; hence the max root can be calculated according to Equation (4). Therefore Lemma 1 is proved.

Many Single-root Nettrees with depth m are used to solve the problem for a pattern with length m. If the root of a Single-root Nettree is given, we can calculate the range of the m-th level leaves according to the length and gap constraints.

Definition 10. The minimal leaf and maximal leaf on the m-th level in a Single-root Nettree with root n_1^r are known as the min leaf and max leaf and are denoted by $MinLf(n_1^r)$ and $MaxLf(n_1^r)$, respectively.

Lemma 2. $MinLf(n_1^r)$ and $MaxLf(n_1^r)$ can be calculated via the following equations:

$$MinLf(n_1^r) = \max(r + MinLen - 1, r + m - 1 + \sum_{k=0}^{m-2} a_k) \tag{5}$$

$$MaxLf(n_1^r) = \min(r + MaxLen - 1, r + m - 1 + \sum_{k=0}^{m-2} b_k,\ n - 1) \tag{6}$$

Proof: If the beginning position is r, the minimal ending position is $r + MinLen$-1, according to the length constraints. Similarly, with respect to the gap constraints, another lower bound of $MinLf(n_1^r)$ is $r + m$-1 + $\sum_{k=0}^{m-2} a_k$. So the min leaf with root n_1^r can be calculated according to Equation (5). When calculating the max leaf of a Single-root Nettree with root n_1^r, not only the length and gap constraints but also the length of the sequence should be considered. The upper bound of $MaxLf(n_1^r)$ should be $r + MaxLen$-1 and $r + m$-1 + $\sum_{k=0}^{m-2} b_k$ according to the length and gap constraints, respectively. Meanwhile, $MaxLf(n_1^r)$ must be no greater than n-1. $MaxLf(n_1^r)$ can therefore be calculated according to Equation (6). Hence Lemma 2 is proved.

Next, we determine the range of node labels on the j-th level in a Single-root Nettree.

Definition 11. The minimal label and maximal label of a node on the j-th level of a Single-root Nettree with root $n_1{}^r$ are known as the min brother and max brother and are denoted by $MinBr(n_1^r, j)$ and $MaxBr(n_1^r, j)$, respectively.

Lemma 3. $MinBr(n_1^r, j)$ and $MaxBr(n_1^r, j)$ $(2 \le j < m)$ can be calculated via the following equations:

$$MinBr(n_1^r, j) = \max(r + j - 1 + \sum_{k=0}^{j-2} a_k, MinLf\left(n_1^r\right) - m + j - \sum_{k=j-1}^{m-2} b_k) \tag{7}$$

$$MaxBr(n_1^r, j) = \min(r + j - 1 + \sum_{k=0}^{j-2} b_k, MinLf\left(n_1^r\right) - m + j - \sum_{k=j-1}^{m-2} a_k) \tag{8}$$

Proof: The j-th level nodes in a Single-root Nettree handle $p_{j\text{-}1}$. We know that the minimal span for sub-pattern $p_0[a_0,b_0]p_1 \ldots [a_{j\text{-}2}, b_{j\text{-}2}]p_{j\text{-}1}$ is $j + \sum_{k=0}^{j-2} a_k$. Therefore, the lower bound of $MinBr(n_1^r, j)$ is $r + j$-1 + $\sum_{k=0}^{j-2} a_k$ from the root r downward to the j-th level, according to the gap constraints. Meanwhile, the maximal span for sub-pattern $p_{j\text{-}1}[a_{j\text{-}1},b_{j\text{-}1}]p_j \ldots [a_{m\text{-}2},b_{m\text{-}2}]p_{m\text{-}1}$ is m-$j + 1 + \sum_{k=j-1}^{m-2} b_k$. When we know that the min leaf is $MinLf(n_1^r)$, the lower bound of $MinBr(n_1^r, j)$ is $MinLf(n_1^r)$ -$m + j$-$\sum_{k=j-1}^{m-2} b_k$, from the min leaf upward to the j-th level. So $MinBr(n_1^r, j)$ can be calculated according to Equation (7). Similarly, it is easy to show the correctness of Equation (8). Hence Lemma 3 is proved.

Finally, we calculate the range of children for each node, thereby enabling the creation of a Single-root Nettree.

Definition 12. The minimal child and maximal child of node n_j^i are known as the min child and max child and are denoted by $MinChd(n_j^i)$ and $MaxChd(n_j^i)$, respectively.

Lemma 4. $MinChd(n_j^i)$ and $MaxChd(n_j^i)$ $(1 \le j < \mathrm{m})$ can be calculated via the following equations:

$$MinChd(n_1^r, j) = \max(r + 1 + a_j - 1, MinBr\left(n_1^r, j + 1\right)) \tag{9}$$

$$\begin{aligned} MaxChd(n_1^r, j) &= \min(r + 1 + b_j - 1, MaxBr\left(n_1^r, j + 1\right)) \\ MaxChd(n_1^r, j) &= \min(r + 1 + b_j - 1, MaxBr\left(n_1^r, j + 1\right)) \end{aligned} \tag{10}$$

Proof: It is easy to see that the first child of n_j^i is $i + 1 + a_{j\text{-}1}$ according to the j-th group of gap constraints. However, the first child is also no less than $MinBr\left(n_1^r, j + 1\right)$, so $MinChd(n_j^i)$ can be calculated according to Equation (9). Similarly, it is easy to show the correctness of Equation (10). Hence Lemma 4 is proved.

Example 3. Given P=a[0,2]g[1,3]a and S=atggaga, which are the same P and S as in Example 1. In this example, $MinLen$ and $MaxLen$ are 4 and 6, respectively. We will create a Single-root Nettree with root 0 using the above concepts and lemmas.

According to the pattern and sequence, we know that a_0=0, b_0=2, a_1=1, b_1=3, m=3 and n=7.

Step 1. Since the root is 0, the Single-root Nettree has only one root: n_1^0.

Step 2. We determine the leaves of the Single-root Nettree. According to Equation (5), when r is 0, we can calculate that $0 + 4$-1=3 and $0 + 3$-$1 + 0 + 1$=3. So, $MinLf(n_1^0) = \max(3, 3) = 3$. Similarly, $MaxLf(n_1^0) = \min(0 + 6 - 1, 7 - 1, 0 + 3 - 1 + 2 + 3) = 5$. Therefore, the Single-root Nettree has three leaves: n_3^3,n_3^4 and n_3^5.

Step 3. Now we determine the min brother and max brother for each level. Since m is 3, we only need to calculate the min brother and max brother on the second level. According to Equations (7) and (8), $MinBr(n_1^0, 2) = \max(0 + 2 - 1 + 0, 3 - 3 + 2 - 3) = 1$ and $MaxBr(n_1^0, 2) = \min(0 + 2 - 1 + 2, 5 - 3 + 2 - 1) = 3$. Therefore, n_2^1,n_2^2 and n_2^3 are the three nodes on the second level.

Step 4. We calculate the children for each node. The min child of node n_1^0 is max(0 + 1 + 0,1)=1, according to Equation (9). The max child of node n_1^0 is min(0 + 1 + 2,3)=3, according to Equation (10). So, root n_1^0 has three children:

n_2^1, n_2^2 and n_2^3. Similarly, we know that node n_2^1 has three children: n_3^3, n_3^4 and n_3^5; node n_2^2 has two children: n_3^4 and n_3^5; and node n_2^3 has one child: n_3^5. The Single-root Nettree is shown in Figure 1.

According to Lemmas 1, 2 and 3, each node in a Single-root Nettree with root n_1^r is subject to the length constraints. Supposing a root-leaf path from root n_1^r to leaf $n_m^{i_{m-1}}$ is $< n_1^r, n_2^{i_1}, \ldots, n_m^{i_{m-1}} >$, we know that $a_{j\text{-}1} \le i_j\text{-}i_{j\text{-}1}\text{-}1 \le b_{j\text{-}1}$ $(0 < j \le m)$ according to Lemma 4. Therefore, path $< n_1^r, n_2^{i_1}, \ldots, n_m^{i_{m-1}} >$ can be an occurrence, if $D(p_0p_1\ldots p_{m\text{-}1}, s_r s_{i_1}\ldots s_{i_{m-1}})$ is no greater than d. Next we handle the similarity constraint in a Single-root Nettree. While creating the children of n_{j-1}^k, we calculate the similar branch number and the similar number for each child according to $N_b(n_1^r,n_{j-1}^k,j\text{-}1,t)$ and $N_s(n_1^r,n_{j-1}^k,j\text{-}1)$. We also define that the initial values of $N_b(n_1^r,n_j^i,j,t)$ and $N_s(n_1^r,n_j^i,j)$ are all 0.

Lemma 5. Supposing n_j^i is a child of n_{j-1}^k $N_b(n_1^r,n_j^i,j,t)$ and $N_s(n_1^r,n_j^i,j)$ can be updated according to following equations:

$$N_b(n_1^r, n_j^i, j, t) = \begin{cases} N_b\left(n_1^r, n_j^i, j, t\right) + N_b\left(n_1^r, n_{j-1}^k, j-1, t\right) & if\ p_j = s_i \quad (\forall t : 0 \le t \le d) \\ N_b\left(n_1^r, n_j^i, j, t\right) + N_b\left(n_1^r, n_{j-1}^k, j-1, t-1\right) & if\ p_j \ne s_i \quad (\forall t : 1 \le t \le d) \end{cases} \tag{11}$$

$$N_s(n_1^r, n_j^i, j) = \begin{cases} N_s\left(n_1^r, n_j^i, j\right) + N_s\left(n_1^r, n_{j-1}^k, j-1\right) & if\ p_j = s_i \\ N_s\left(n_1^r, n_j^i, j\right) + N_s\left(n_1^r, n_{j-1}^k, j-1\right) - N_b(n_1^r, n_{j-1}^k, j-1, d) & if\ p_j \ne s_i \end{cases} \tag{12}$$

Proof: If p_j is equal to s_i, $N_b\left(n_1^r, n_j^i, j, t\right)$ is the sum of the number of all paths from n_1^r to n_j^i by n_{j-1}^k, with length j-1 and distance t. Therefore, when calculating $N_b\left(n_1^r, n_j^i, j, t\right)$ and $N_s(n_1^r, n_j^i, j)$ of n_{j-1}^k's children, $N_b\left(n_1^r, n_j^i, j, t\right)$ and $N_s(n_1^r, n_j^i, j)$ can be updated according to $N_b\left(n_1^r, n_j^i, j, t\right) + N_b\left(n_1^r, n_{j-1}^k, j-1, t\right)$ and $N_s\left(n_1^r, n_j^i, j\right) + N_s\left(n_1^r, n_{j-1}^k, j-1\right)$, respectively.

If p_j is not equal to s_i, $N_b\left(n_1^r, n_j^i, j, t\right)$ is the sum of the number of all paths from n_1^r to n_j^i by n_{j-1}^k, with length j-1 and distance t-1. Therefore, $N_b\left(n_1^r, n_j^i, j, t\right)$ can be updated according to $N_b\left(n_1^r, n_j^i, j, t\right) + N_b\left(n_1^r, n_{j-1}^k, j-1, t-1\right)$. Since $N_s\left(n_1^r, n_{j-1}^k, j-1\right)$ is equal to $\sum_{t=0}^{d} N_b\left(n_1^r, n_{j-1}^k, j-1, t\right)$, $\sum_{t=0}^{d-1} N_b\left(n_1^r, n_{j-1}^k, j-1, t\right)$ is $N_s\left(n_1^r, n_{j-1}^k, j-1\right) - N_b(n_1^r, n_{j-1}^k, j-1, d)$. Therefore, $N_s(n_1^r, n_j^i, j)$ will be updated according to $N_s\left(n_1^r, n_j^i, j\right) + N_s\left(n_1^r, n_{j-1}^k, j-1\right) - N_b(n_1^r, n_{j-1}^k, j-1, d)$. Therefore Lemma 5 is proved.

According to Lemma 5, we can calculate the similar number of each node level by level, thereby revealing the similar number of each leaf. Next we calculate the number of root-leaf paths of a Single-root Nettree that meet the similarity constraint.

Definition 13. The number of paths from root n_1^r to all its leaves subject to the similarity constraint is known as the root solution of n_1^r and is denoted by $R_s\left(n_1^r\right)$.

Lemma 6. $R_s\left(n_1^r\right)$ can be calculated via the following equation:

$$R_s\left(n_1^r\right) = \sum_{h=MinLf(n_1^r)}^{MaxLf(n_1^r)} N_s(n_1^r, n_m^h, m) \tag{13}$$

Proof: Supposing n_m^h is the min leaf of a Single-root Nettree with root n_1^r, we know that $N_s(n_1^r, n_m^h, m)$ is the similar number from root n_1^r to node n_m^h with length m, with h varying from $MinLf(n_1^r)$ to $MaxLf(n_1^r)$. $R_s\left(n_1^r\right)$ can thus be calculated according to Equation (13). Hence Lemma 6 is proved.

Finally, we use the root solution of each Single-root Nettree to compute the solution to the problem.

Lemma 7. $N(P,S,d)$ can be calculated by the following equation.

$$N(P, S, d) = \sum_{r=0}^{MaxRt} R_s(n_1^r) \tag{14}$$

Proof: The solution to the problem is the sum of $R_s(n_1^r)$ for r from 0 to $MaxRt$, since $R_s(n_1^r)$ is the number of paths from root n_1^r to all its leaves subject to the similarity constraint. Therefore, $N(P,S,d)$ can be calculated according to Equation (14).

Theorem 1. If $N_s(n_1^r,n_j^{i_{j-1}},j)$ is 0, then node $n_j^{i_{j-1}}$ has no child, which means that we can remove the sub-Nettree whose root is node $n_j^{i_{j-1}}$.

Proof: Supposing a path from root n_1^r to node $n_k^{i_{k-1}}$ by node $n_j^{i_{j-1}}$ is $<n_1^r,n_2^i,\ldots,n_j^{i_{j-1}},\ldots,n_k^{i_{k-1}}>$, it is easy to see that $D(p_0p_1\ldots p_{j\text{-}1}, s_rs_{i_1}\ldots s_{i_{j-1}})$ is no greater than $D(p_0p_1\ldots p_{k\text{-}1}, s_rs_{i_1}\ldots s_{i_{k-1}})$, i.e. $D(p_0p_1\ldots p_{j\text{-}1}, s_rs_{i_1}\ldots s_{i_{j-1}}) \le D(p_0p_1\ldots p_{k\text{-}1}, s_rs_{i_1}\ldots s_{i_{k-1}})$. If $N_s(n_1^r,n_j^{i_{j-1}}, j)$ is 0, the distance between path $<n_1^r,n_2^i,\ldots,n_j^{i_{j-1}}>$ and its corresponding sub-pattern is greater than d, i.e. $D(p_0p_1\ldots p_{j\text{-}1}, s_rs_{i_1}\ldots s_{i_{j-1}}) > d$. So, $D(p_0p_1\ldots p_{j\text{-}1}\ldots p_{k\text{-}1}, s_rs_{i_1}\ldots s_{i_{j-1}}\ldots s_{i_{k-1}})$ is greater than d and $N_s(n_1^r, n_k^{i_{k-1}},k)$ will be 0. Therefore, the deletion of node $n_k^{i_{k-1}}$ will not affect the similar number of its leaves. So, node $n_j^{i_{j-1}}$ has no child. Hence, we can safely delete the sub-Nettree whose root is node $n_j^{i_{j-1}}$. This completes the proof.

4.3. SONG algorithm

Based on these concepts and lemmas, the main framework of SONG is given in Algorithm 1.

SONG algorithm

Input: $P=p_0[a_0, b_0]p_1\ldots[a_{j\text{-}1}, b_{j\text{-}1}]p_j\ldots[a_{m\text{-}2}, b_{m\text{-}2}]p_{m\text{-}1}$, $S=s_0s_1\ldots s_i\ldots s_{n\text{-}1}$, *MinLen*, *MaxLen* and d
Output: $N(P,S,d)$

```
1: for (r=0; r ≤ MaxRt; r++)
2:     Create root n_1^r and initialize N_b(n_1^r,n_1^r,1,d) and N_s(n_1^r,n_1^r,1) according to Definition 6;
3:     Compute MinLf(n_1^r) and MaxLf(n_1^r) according to Equations (5) and (6), respectively;
4:     for (j=2; j ≤ m; j++)
5:         Compute MinBr(n_1^r,j) and MaxBr(n_1^r,j) according to Equations (7) and (8), respectively;
6:         Initialise the j-th level nodes according to MinBr(n_1^r,j) and MaxBr(n_1^r,j), respectively;
7:         for (u = 0; u < MaxBr(n_1^r,j − 1) − MinBr(n_1^r,j − 1) + 1; u++)
8:             k=get the u-th node on the (j-1)-th level;
9:             if N_s(n_1^r, n_{j-1}^k,j)=0 continue; // according to Theorem 1
10:            Compute MinChd(n_{j-1}^k) and MaxChd(n_{j-1}^k) according to Equations (9) and (10), respectively;
11:            for (i=MinChd(n_{j-1}^k); i ≤ MaxChd(n_{j-1}^k); i++)
12:                Update N_b(n_1^r,n_j^i,j,d) and N_s(n_1^r,n_j^i,j) according to Equations (11) and (12), respectively;
13:            end for
14:        end for
15:    end for
16:    Compute R_s(n_1^r) according to Equation (13);
17: end for
18: Compute N(P,S,d) according to Equation (14).
```

4.4. Analysis of SONG

Theorem 2. The space complexity of SONG is $O(m^2*W*d)$, where m, W and d are the pattern length, the maximal gap and the similarity constraint, respectively.

Proof: Single-root Nettrees are created one after another; no more than one Single-root Nettree exists in the memory at any time, as shown in SONG. As a Single-root Nettree has no more than $(j\text{-}1)*W$ nodes on the j-th level ($1 < j \le m$), it has no more than $O(m^2*W)$ nodes. Therefore, the space complexity of SONG is $O(m^2*W*d)$, because a node has $d+1$ similar branch numbers and 1 similar number.

Theorem 3. The time complexity of SONG is $O((n\text{-}m)*m^2*W^2*d)$.

Proof: The time complexity analysis of SONG is given as follows: The number of iterations of line 1 is $O(n\text{-}m)$, because there are no more than $n\text{-}m+1$ Single-root Nettrees which need to be created. Line 2 costs a constant time. The time complexity of line 3 is $O(m)$. The number of iterations of line 4 is $O(m)$. The time complexity of line 5 is $O(m)$. The time complexity of line 6 is $O(m*W*d)$, since a level has no more than $O(m*W)$ nodes. The number of iterations of line 7 is $O(m*W)$. Lines 8, 9, and 10 cost a constant time. The number of iterations of line 11 is $O(W)$, since one node has

no more than W children. The time complexity of line 12 is $O(d)$, since t is no more than d. The time complexities of lines 16 and 18 are $O(m*W)$ and $O(n\text{-}m)$, respectively. Therefore, the overall time complexity of SONG is $O((n\text{-}m)*m^2*W^2*d)$.

4.5. A running example

Example 4. Here we use P=a[0,2]g[1,3]a, S=atggaga, $MinLen$=4, $MaxLen$=6 and d = 1 to show how SONG works.

The Single-root Nettree whose root is 0 was shown earlier in Figure 1. Since s_0='a'=p_0='a', we know that $N_s(n_1^0n_1^0,n_1^0,1)$=1, $N_b(n_1^0,n_1^0,1,0)$=1 and $N_b(n_1^0,n_1^0,1,1)$=0 according to line 2 of SONG. We know that node n_1^0 has three children: n_2^1, n_2^2 and n_2^3. Since s_1='t'$\neq p_1$='g', $N_s(n_1^0,n_2^1,2)$=1, $N_b(n_1^0,n_2^1,2,0)$=0 and $N_b(n_1^0,n_2^1,2,1)$=1 according to line 12 of SONG. Similarly, we know that $N_s(n_1^0,n_2^2,2)$=1, $N_b(n_1^0,n_2^2,2,0)$=1, $N_b(n_1^0,n_2^2,2,1)$=0, $N_s(n_1^0,n_2^3,2)$=1, $N_b(n_1^0,n_2^3,2,0)$=1 and $N_b(n_1^0,n_2^3,2,1)$=0.

Now we calculate the similar number and similar branch number of node n_3^4, which has two parents: nodes n_2^1 and n_2^2. When we update the similar number and similar branch number of child node n_3^4 according to its parent, node n_2^1, we know that $N_s(n_1^0,n_3^4,3)=N_s(n_1^0,n_3^4,3)+N_s(n_1^0,n_2^1,2)$=1, $N_b(n_1^0,n_3^4,3,0)=N_b(n_1^0,n_3^4,3,0)+N_b(n_1^0,n_2^1,2,0)$=0 and $N_b(n_1^0,n_3^4,3,1)=N_b(n_1^0,n_3^4,3,1)+N_b(n_1^0,n_2^1,2,1)$=1, since s_4='a'=p_2='a'. Similarly, we update the similar number and similar branch number of child node n_3^4 according to its other parent, node n_2^2. $N_s(n_1^0,n_3^4,3)$, $N_b(n_1^0,n_3^4,3,0)$ and $N_b(n_1^0,n_3^4,3,1)$ are $1+1$=2, $0+1$=1 and $1+0$=1, respectively.

Therefore, we can calculate the similar number and similar branch number for each node. Figure 2(a) shows the Single-root Nettree with root n_1^0. On the top of each node there are three numbers: the similar number of node n_j^i and the similar branch number with t=0 and t=1 of node n_j^i. For example, 2,{0,2} on the top of node n_3^5 represent $N_s(n_1^0,n_3^5,3)$ and $\{N_b(n_1^0,n_3^5,3,0), N_b(n_1^0,n_3^5,3,1)\}$, respectively. We can calculate that $R_s(n_1^0)$ is $0+2+2$=4 according to line 16 of SONG.

Similarly, we can create three Single-root Nettrees with roots n_1^1, n_1^2 and n_1^3, as shown in Figure 2b, c and d, respectively. All nodes n_2^4 in Single-root Nettrees with roots n_1^1, n_1^2 and n_1^3 have no child according to Theorem 1. SONG is thus an effective algorithm, since it employs an effective pruning strategy. According to line 16 of SONG, we can easily calculate $R_s(n_1^1)$, $R_s(n_1^2)$ and $R_s(n_1^3)$ as $1+0+2$=3, $0+1$=1 and 0, respectively. Hence, $N(P,S,d)$ is $4+3+1+0$=8 according to line 18 of SONG.

We can enumerate all the eight approximate occurrences as $<0,1,4>$, $<0,2,4>$, $<0,2,5>$, $<0,3,5>$, $<1,2,4>$, $<1,2,6>$, $<1,3,6>$ and $<2,3,6>$, thereby verifying the correctness of the SONG algorithm.

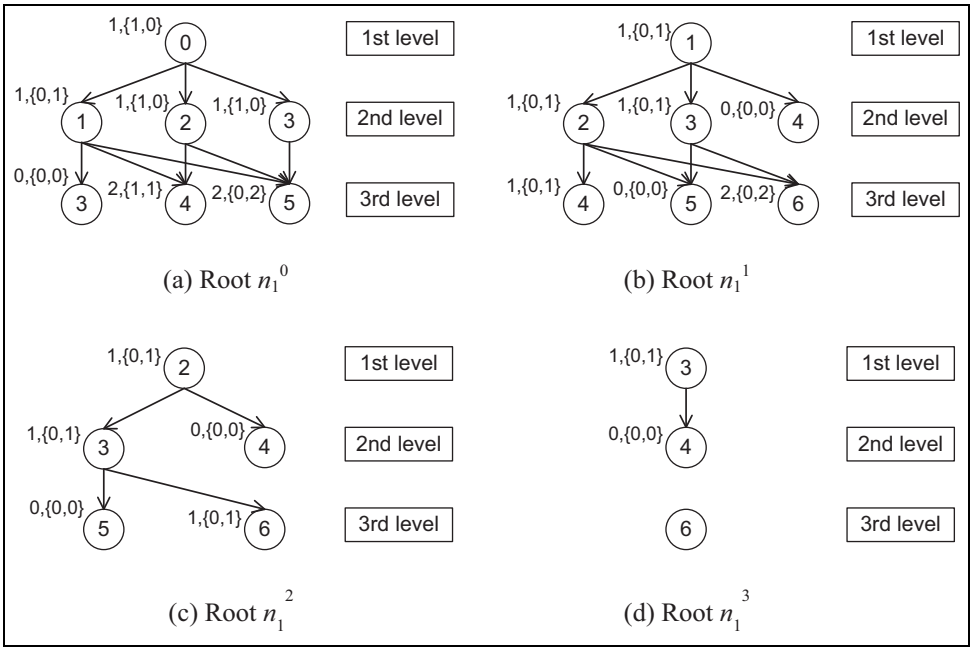

Figure 2. The Single-root Nettrees.

Table 2. Segments of the H1N1 virus sequence.

Sequence	Segment No.	Locus	Length
*S*1	Segment1	CY058563	2286
*S*2	Segment2	CY058562	2299
*S*3	Segment3	CY058561	2169
*S*4	Segment4	CY058556	1720
*S*5	Segment5	CY058559	1516
*S*6	Segment6	CY058558	1418
*S*7	Segment7	CY058557	982
*S*8	Segment8	CY058560	844

Table 3. Patterns.

Name	Pattern
*P*1	a[0,3]t[0,3]a[0,3]t[0,3]a[0,3]t[0,3]a[0,3]t[0,3]a[0,3]t[0,3]a[0,3]t[0,3]a
*P*2	g[1,5]t[0,6]a[2,7]g[3,9]t[2,5]a[4,9]g[1,8]t[2,9]a
*P*3	g[1,9]t[1,9]a[1,9]g[1,9]t[1,9]a[1,9]g[1,9]t[1,9]a[1,9]g[1,9]t[1,9]a[1,9]g[1,9]t
*P*4	g[1,5]t[0,6]a[2,7]g[3,9]t[2,5]a[4,9]g[1,8]t[2,9]a[1,9]g[1,9]t
*P*5	a[0,10]a[0,10]t[0,10]c[0,10]g[0,10]g
*P*6	a[0,5]t[0,7]c[0,9]g[0,11]g
*P*7	a[0,5]t[0,7]c[0,6]g[0,8]t[0,7]c[0,9]g
*P*8	a[5,6]c[4,7]g[3,8]t[2,8]a[1,7]c[0,9]g
*P*9	c[0,5]t[0,5]g[0,5]a[0,5]a

According to loose pattern matching there are only three occurrences: 4, 5 and 6. This example therefore also demonstrates that loose pattern matching is far easier than strict pattern matching.

5. Experimental results and analysis

5.1. Experimental environment and data

In this section, we test our algorithms on real biological data. All experiments were run on a laptop with an Intel(R) Core(TM)2 Duo CPU T7100 1.80GHz and 1.0 GB of RAM, in Windows XP SP 2. VC 6.0 was used to develop all the algorithms. To verify the performances of the proposed algorithm, eight sequences are real biological sequences provided by the National Centre for Biotechnology Information website (http://www.ncbi.nlm.nih.gov/genomes/FLU/SwineFlu.html) (Table 2) and nine sequences are selected from the previous literatures [13, 28] (Table 3).

5.2. Experimental results of theoretical analysis

5.2.1. Relationship. To illustrate the relationship between $N(P,S,d)$ and $M(P,S,t)$ $(0 \leq t \leq d)$, we select sequences $S1$ to $S4$, pattern $Q1$=a[0,2]g[1,3]a and d=1. According to the problem analysis outlined in Subsection 3.2, we know that $N(Q1,S,1)=M(Q1,S,0)+M(Q1,S,1)$. Since the lengths of $Q1$ and λ are 3 and 4, respectively, there are $(\lambda\text{-}1)*m=3*3=9$ patterns whose distances to $Q1$ are 1: $Q2$=c[0,2]g[1,3]a, $Q3$=g[0,2]g[1,3]a, $Q4$=t[0,2]g[1,3]a, $Q5$=a[0,2]a[1,3]a, $Q6$=a[0,2]c[1,3]a, $Q7$=a[0,2]t[1,3]a, $Q8$=a[0,2]g[1,3]c, $Q9$=a[0,2]g[1,3]g and $Q10$=a[0,2]g[1,3]t. We know that when d is 0, $N(P,S,0)$ can be solved using PAIG [13]. Table 4 thus presents the results of $N(P,S,0)$ according to PAIG.

Using SONG, we know that $N(Q1,S1,1)$, $N(Q1,S2,1)$, $N(Q1,S3,1)$ and $N(Q1,S4,1)$ are 4782, 4799, 4248 and 3532, respectively, with the sums of the results for patterns $Q1$ to $Q10$ in sequences $S1$, $S2$, $S3$ and $S4$ also 4782, 4799, 4248 and 3532, respectively. This therefore validates the correctness of $N(P,S,1)=M(P,S,0)+M(P,S,1)$, no matter what the sequences are. These experiments thus verify that an instance in the present study can be transformed into many instances of [13], thereby also verifying the correctness of SONG.

Table 4. The results of $N(P,S,0)$ according to PAIG.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Sum
S1	682	286	497	432	774	392	490	341	485	403	4782
S2	608	249	401	400	1006	501	530	330	366	408	4799
S3	556	243	410	436	659	393	490	353	383	325	4248
S4	460	197	288	356	658	323	499	198	263	290	3532

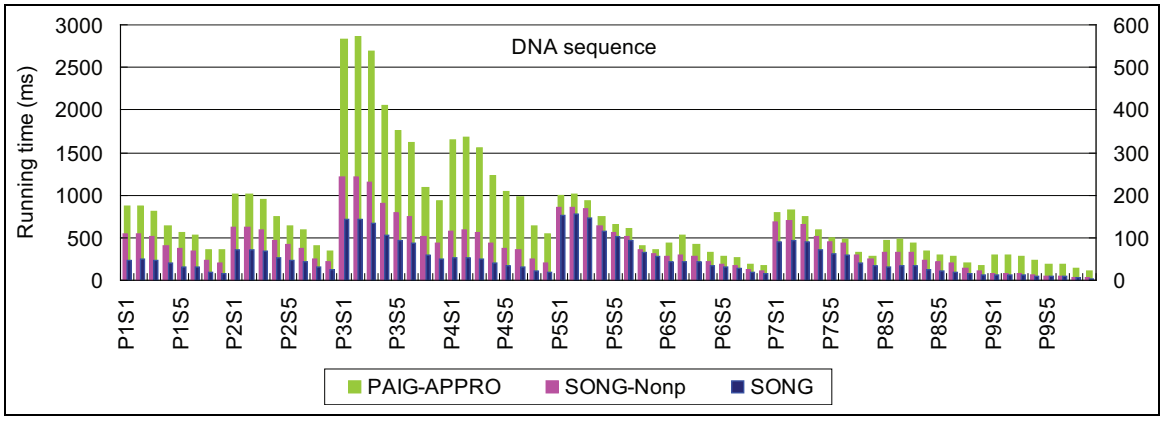

Figure 3. Comparison of algorithms running time.

5.2.2. Performances. We design an algorithm named PAIG-APPRO based on PAIG [13], using the dynamic programming principle. PAIG adopts a three-dimensional array, with the time complexity and space complexity being $O(n*m^2*W^2)$ and $O(n*m*W)$, respectively. Unlike PAIG [13], PAIG-APPRO has to deal with the similarity constraint and thus uses a four-dimensional array; the time complexity and space complexity of PAIG-APPRO are therefore $O(n*m^2*W^2*d)$ and $O(n*m*W*d)$, respectively. In order to verify the effectiveness of SONG pruning strategy, we also designed an algorithm, SONG-Nonp, which deletes line 9 of SONG. PAIG-APPRO, SONG-Nonp and SONG can all be downloaded from http://wuc.scse.hebut.edu.cn/song/index.htm. In order to demonstrate the performances of the three algorithms, we use 72 instances: patterns from $P1$ to $P9$ and sequences from $S1$ to $S8$. In these instances, *MinLen*, *MaxLen* and d are 20, 40 and 2, respectively. Since the three algorithms are all completeness algorithms, we omit the results and only compare the running time of the 72 instances (as shown in Figure 3). It is worth noting that the running time of PAIG-APPRO refers to the time data on the left and the other algorithms refer to the right.

From Figure 3, we know that SONG is more effective than other competitive algorithms. For instance, the running time of PAIG-APPRO, SONG-Nonp and SONG for $P1$ in $S1$ are about 828 ms, 112 ms and 53 ms, respectively. SONG is thus about 16 times faster than PAIG-APPRO for this instance and about two times faster than SONG-Nonp. To solve all 72 instances, PAIG-APPRO, SONG-Nonp and SONG take 53 s, 6 s and 3.7 s, respectively. Therefore, SONG is about 14 times and 1.6 times faster than PAIG-APPRO and SONG-Nonp for all 72 instances, respectively. Although SONG, SONG-Nonp and PAIG-APPRO have the same time complexity, the complexity of SONG is its upper bound. SONG is a more effective algorithm than SONG-Nonp, since SONG employs an effective pruning strategy which is given in Theorem 1. Therefore, some sub-Nettrees can be removed from SONG. SONG is also faster than PAIG-APPRO, since SONG only calculates the values of non-zero nodes, while PAIG-APPRO calculates on a four-dimensional array in which most values are zero.

5.2.3. Binomial distribution. In order to explore $M(P,S,d)/N(P,S,m)$ distributions of the patterns in real biological sequences, we match patterns $P1$, $P2$, $P3$ and $P4$ in the corresponding biological sequences $S1$, $S2$, $S3$ and $S4$; the respective comparisons between the practical and theoretical results for $M(P,S,d)/N(P,S,m)$ are displayed in Figure 3a, b, c and d. Here the theoretical ratio, abbreviated as TR, follows the binomial distribution $B(m,1\text{-}U(a)*V(a)\text{-}U(t)*V(t)\text{-}U(c)*V(c)\text{-}U(g)*V(g))$, in which $U(a)$ and $V(a)$ are the probabilities of 'a' in the pattern and sequence, respectively, as well as the other three characters. Length constraints are neglected in order to reflect the natural distributions.

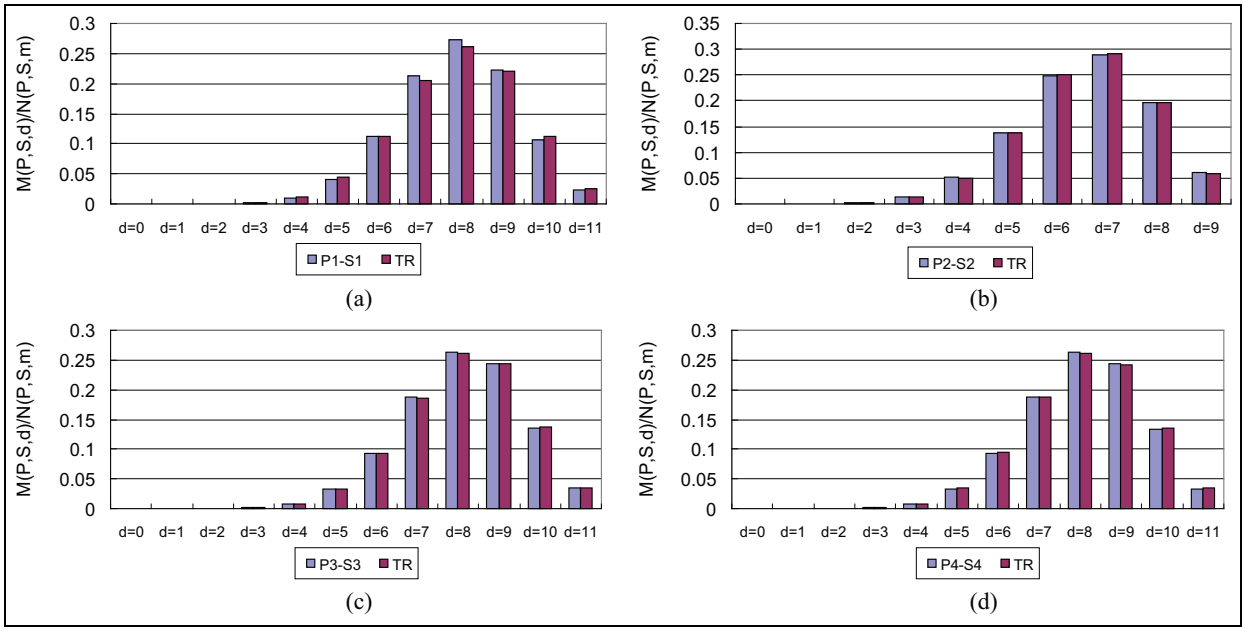

Figure 4. Comparison between practical and theoretical $M(P,S,d)/N(P,S,m)$ for four different patterns in four sequences.

Table 5. Variation in $N(P,S,d)$ with increasing values of d.

	d=0	d=1	d=2	d=3	d=4	d=5	d=6
S1	23397	718175	9283388	68215198	321073601	1026305321	2311005598
S2	47546	1088973	11665944	76552765	337558762	1041566230	2308607725
S3	28722	765497	9197628	64998756	300404923	956184272	2160599453
S4	25691	644831	7611195	53585581	246565547	778125582	1739500658

From Figure 4, we can observe that whereas the practical and theoretical ratios of $M(P,S,d)/ N(P,S,m)$ are very similar in Figure 4b, c and d, in Figure 4a the practical ratio is obviously a little higher than the theoretical ratio when d=8. Nevertheless, this discrepancy is still about 1% within the allowable error range. As the practical and theoretical ratios are generally equal for all values of d, $M(P,S,d)/N(P,S,m)$ therefore approximately follows the binomial distribution $B(m,1\text{-}U(a)*V(a)\text{-}U(t)*V(t)\text{-}U(c)*V(c)\text{-}U(g)*V(g))$ in the biological sequences. Furthermore, by comparing $P3$ and $P4$ with Figure c and d, we can conclude that the gaps in the pattern have no significant impact on $M(P,S,d)/N(P,S,m)$ following the binomial distribution $B(m,1\text{-}U(a)*V(a)\text{-}U(t)*V(t)\text{-}U(c)*V(c)\text{-}U(g)*V(g))$. This is due to the fact that 'a', 't', 'c' and 'g' are distributed randomly in the selected sequences.

According to this law, if we know $M(P,S,0)$, then $M(P,S,d)$ and $N(P,S,d)$ $(0 < d \leq m)$ can be estimated easily. For example, the length of $P3$ and $M(P3,S,0)$ are 11 and 2,991,637, respectively. We can estimate that $M(P3,S,3)$ and $N(P3,S,11)$ are about 1.09*10^10 and 7.27*10^12, respectively. Actually, $M(P3,S,3)$ and $N(P3,S,11)$ are 1.06*10^10 and 7.35*10^12, respectively.

5.2.4. Similarity constraint evaluation. In order to illustrate how d affects $N(P,S,d)$ and the running time of SONG, we here use pattern $P2$ and sequences $S1$ to $S4$. The changes in $N(P,S,d)$ and the running time of SONG with increasing values of d (from 0 to 6) are shown in Table 5 and Figure 5, respectively.

From Table 5, we know that $N(P,S,d)$ rises rapidly when d increases, no matter what the sequences are. According to the problem definition, we know that $M(P,S,t)$ can be expressed by $C_m^t * (\lambda - 1)^t$ patterns whose distances to P are t; $M(P,S,t)$ is thus exponential with t. Accordingly, as $N(P,S,d)$ is the sum of $M(P,S,t)$ $(0 \leq t \leq d)$, $N(P,S,d)$ grows rapidly when d increases.

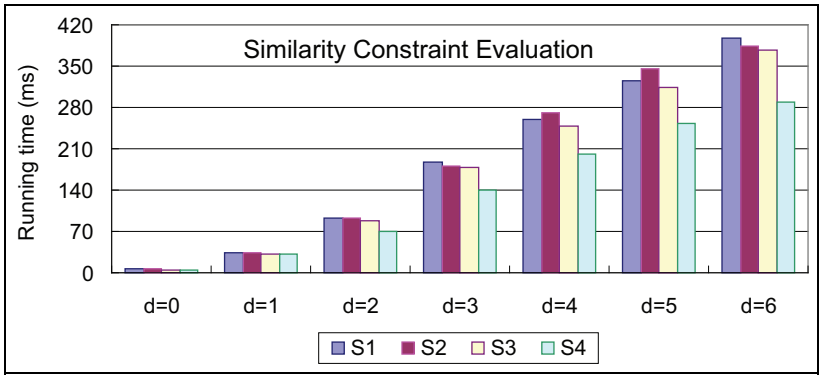

Figure 5. Variation in the running time of SONG increasing values of *d*.

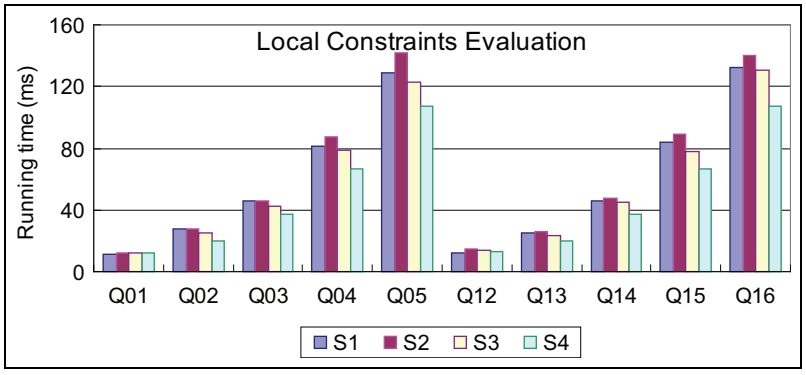

Figure 6. Variation in the running time of SONG with different a_j and b_j

Table 6. Variation in $N(P,S,d)$ with different a_j and b_j.

	Q01	Q02	Q03	Q04	Q05	Q12	Q13	Q14	Q15	Q16
S1	327	22687	456913	5103735	36706713	405	31627	800307	7706700	42511798
S2	442	33988	542888	5616088	42037304	860	31741	685563	8205722	53559645
S3	250	26682	443385	4209139	30561934	702	30676	577601	6296132	40895241
S4	288	32055	659353	7888908	51679713	367	43002	986975	9692799	63947998

However, as a whole, the running time of SONG increases almost linearly by d according to Figure 5, because the time complexity of SONG is $O((n\text{-}m)*m^2*W^2*d)$. Specifically, when d increases from 0 to 3, the running time growth rate is subject to a clear increase, whereas this growth rate is much less obvious when d increases from 3 to 6. This discrepancy is due to the fact that Theorem 1 is used to remove certain sub-Nettrees and the bigger d becomes, the smaller the removed sub-Nettrees tend to be. Thus, Theorem 1 affects the running time of SONG to a lesser degree when d is close to the pattern length. The above experiments therefore validate the correctness of SONG time complexity variation with d.

5.2.5. Local constraint evaluation. Here we show how W affects $N(P,S,d)$ and the running time of SONG. In this part, we use 10 patterns which have same characters with pattern $P1$ and different gap constraints. These patterns are called Qxy, where x and y are the min gap and max gap, respectively. For instance, $Q12$ is a[1,2]t[1,2]a[1,2]t[1,2]a[1,2]t[1,2]a[1,2]t[1,2]a[1,2]t[1,2]a. The changes in $N(P,S,d)$ and the running time of SONG with different W are shown in Table 6 and Figure 6, respectively.

Table 7. The results of $N(P,S,d)$ with increasing values of m.

	m=3	m=4	m=5	m=6	m=7	m=8	m=9	m=10
S1	24202	44322	81499	110217	173004	209553	297309	337128
S2	25107	48543	94055	127811	214381	244166	371856	377051
S3	22734	42802	78702	111994	175387	210799	302521	317632
S4	18757	37477	74823	112230	195905	243996	400642	441417

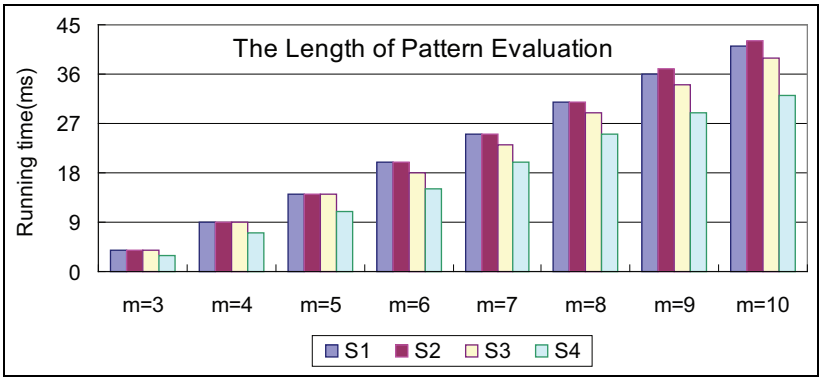

Figure 7. Variation in the running time of SONG with increasing values of m.

From Table 6, we know that $N(P,S,d)$ grows very fast when W increases. According to Zhang et al. [16], it is easy to prove that the upper bound of $N(P,S,d)$ is $O((n\text{-}m)*W^{m-1})$. So when W increases, $N(P,S,d)$ grows very fast. Experimental results also verify this phenomenon. According to Figure 6, we know that when W is the same, the running time of SONG is nearly the same, no matter what the min gaps are. For instance, when W is 2, the running time of SONG for patterns $Q02$ and $Q13$ in all four sequences is nearly 25ms. The running time of SONG is approximately proportional to the square of W according to Figure 6, because the time complexity of SONG is $O((n\text{-}m)*m^2*W^2*d)$. So these experiments validate the correctness of the time complexity of SONG with W.

5.2.6. Length of pattern evaluation. We illustrate how m affects $N(P,S,d)$ and the running time of SONG. To avoid other factors affecting the results, we use prefix sub-patterns of pattern $P1$ with length m and sequences from $S1$ to $S4$. For instance, the pattern is 'a[0,3]t[0,3]a[0,3]t[0,3]a' when m is 5. The changes in $N(P,S,d)$ and the running time of SONG with increasing values m (from 3 to 10) are shown in Table 7 and Figure 7, respectively.

From Table 7, the results verify that when m increases, $N(P,S,d)$ will also increase rapidly. Although the time complexity of SONG is $O((n\text{-}m)*m^2*W^2*d)$, from Figure 7, we know that the running time of SONG nearly grows linearly with m. For instance, the running time of SONG for m=3, m=4 and m=5 in sequence $S2$ are close to 5, 10 and 15, respectively. The reason is that $O((n\text{-}m)*m^2*W^2*d)$ is the upper bound of the time complexity of SONG, and according to Theorem 1, some sub-Nettrees can be removed. So in practice, SONG is faster than $O((n\text{-}m)*m^2*W^2*d)$.

5.2.7. Length of sequence evaluation. Subsequences of $S1$ with lengths of 300, 600, 900, 1200, 1500 and 2100 and patterns from $P1$ to $P4$ are used to illustrate how n affects the changes in $N(P,S,d)$ and the running time of SONG with increasing values of n which are shown in Table 8 and Figure 8, respectively.

From Table 8, the results verify that when n increases, $N(P,S,d)$ will also increase rapidly. We also notice that $N(P3,S,d)$ is far bigger than that of other patterns. The reason is that W of $P3$ is 9 and is bigger than that of other patterns. The results also validate the effect of W that mentioned above. From Figure 8, we know that the running time of SONG is nearly linear growth with n. For instance, the running time of SONG of pattern $P3$ in sequences with lengths n=300, n=600 and n=900 are close to 50 ms, 100 ms and 150 ms, respectively. These experimental results validate the correctness of the time complexity of SONG with n.

Table 8. The results of $N(P,S,d)$ with increasing values of n.

	n=300	n=600	n=900	n=1200	n=1500	n=1800	n=2100
P1	39042	163294	193151	243058	271743	387301	441717
P2	691969	2132130	3627136	4738694	6178700	7561431	8620674
P3	52440283	210952162	418907044	558519702	802285526	1005791483	1169065898
P4	2981125	11210960	21857921	28776454	39328331	49369780	56288411

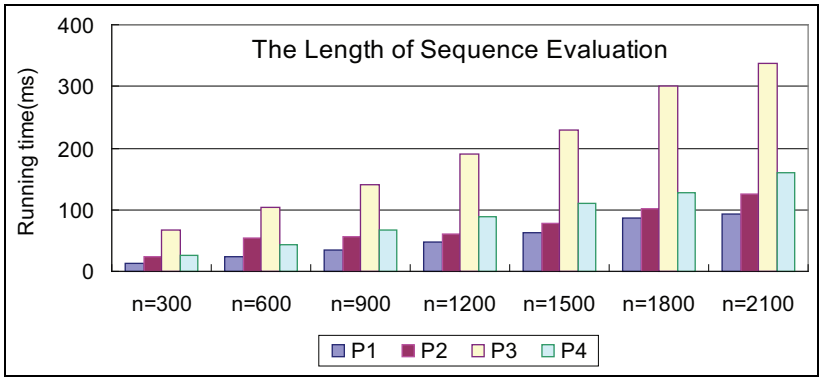

Figure 8. Variation in the running time of SONG with increasing values of n.

Table 9. The results of $N(P,S,d)$ with different *MinLen*.

	MinLen=11	*MinLen*=15	*MinLen*=19	*MinLen*=23	*MinLen*=27	*MinLen*=31	*MinLen*=35
S1	456913	456887	452696	398618	224979	59994	7305
S2	542888	542726	536283	465636	242518	48020	4224
S3	443385	443302	439273	386845	210564	52640	4276
S4	659353	659262	650915	562667	316390	92985	7964

5.2.8. Length constraints evaluation. Here we show how the length constraints affect $N(P,S,d)$ and the running time of SONG. In this part, pattern $P1$ and sequences from $S1$ to $S4$ are used. First, we set *MaxLen* 41, the maximum length of occurrences of pattern $P1$, and use different *MinLen* with 11, 15, 19, 23, 27, 31 and 35 to show how *MinLen* affects the changes which are shown in Table 9 and Figure 9, respectively. Then we set *MinLen* to 11, the minimum length of occurrences of pattern $P1$, and use different *MaxLen* with 14, 18, 22, 26, 30, 34 and 38 to test how *MaxLen* affects the changes which are shown in Table 10 and Figure 10, respectively.

We know that if *MaxLen-MinLen* + 1 decreases, $N(P,S,d)$ will also decrease rapidly, because the less number of leaves and nodes of each Single-root Nettree has, therefore the faster SONG is. We know that if *MinLen* increases, *MaxLen-MinLen* + 1 will decrease. Therefore the less $N(P,S,d)$ is, the faster SONG is. For instance, when *MinLen* is 31, $N(P,S,d)$ in $S1$ is 59994 and the running time of SONG is about 30 ms, while *MinLen* is 35, $N(P,S,d)$ is 7305 and the running time of SONG is about 20 ms. Similar phenomena can also be found in Table 10 and Figure 10. For instance, when *MaxLen* is 18, $N(P,S,d)$ in $S1$ is 4217 and the running time of SONG is about 20 ms, while *MaxLen* is 14, $N(P,S,d)$ is 26 and the running time of SONG is about 10 ms. Therefore these results validate that the smaller *MaxLen-MinLen* + 1 is, the less $N(P,S,d)$ is and the faster SONG is.

From Figure 9, we know that the running time of SONG are almost the same when *MinLen* are 11, 15 and 19. For instance, when *MinLen* are 11, 15 and 19, the running time of SONG for $S1$ is about 43 ms. The reason lies in that when *MinLen* is closed to $m + \sum_{k=0}^{m-2} a_k$ only a small amount of nodes can be deleted because they are not subject to the length constraints. So the running time of SONG is almost the same. Similarly, we can also observe the same phenomenon in Figure 10.

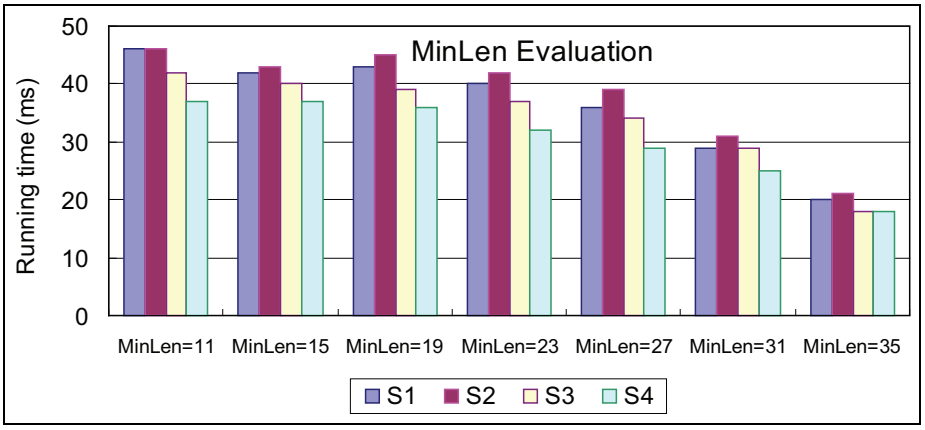

Figure 9. Variation in the running time of SONG with different *MinLen*.

Table 10. The results of $N(P,S,d)$ with different *MaxLen*.

	MaxLen=14	*MaxLen*=18	*MaxLen*=22	*MaxLen*=26	*MaxLen*=30	*MaxLen*=34	*MaxLen*=38
S1	26	4217	58295	231934	396919	449608	456703
S2	162	6605	77252	300370	494868	538664	542828
S3	83	4112	56540	232821	390745	439109	443331
S4	91	8438	96686	342963	566368	651389	659307

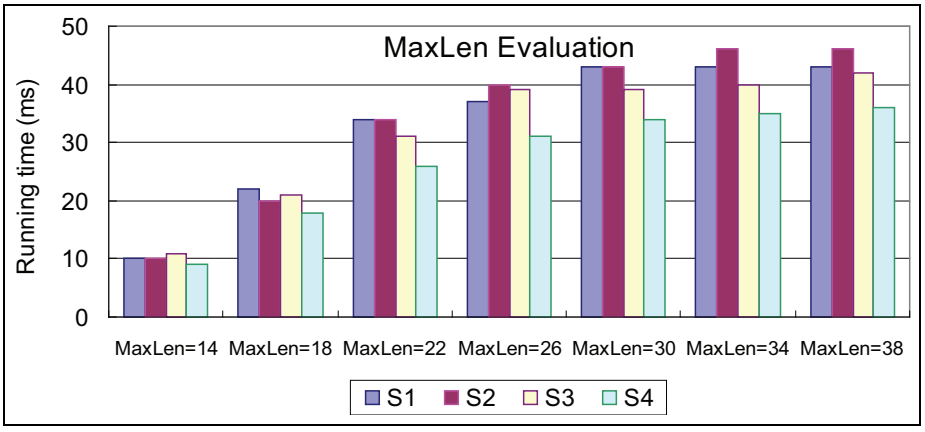

Figure 10. Variation in the running time of SONG with different *MaxLen*.

Hence, in these subsections, all these experiments illustrate how d, W, m, n, *MinLen* and *MaxLen* affect $N(P,S,d)$ and validate the efficiency and correctness of SONG.

5.3. Experimental results on protein

To evaluate the scalability of SONG-Nonp and SONG, we select seven datasets of protein which are ASTRAL95_1_171 (with length 109424) and its sub-sequences with length 15000, 30000, 45000, 60000, 75000 and 90000. ASTRAL95_1_171 is used in [18] and can be downloaded from http://gi.cebitec.uni-bielefeld.de/comet/force/indexOld.html. Nine patterns (from $Q1$ to $Q9$) are g[1,15]t[1,15]a[1,15]a[1,15]t[1,15]a,

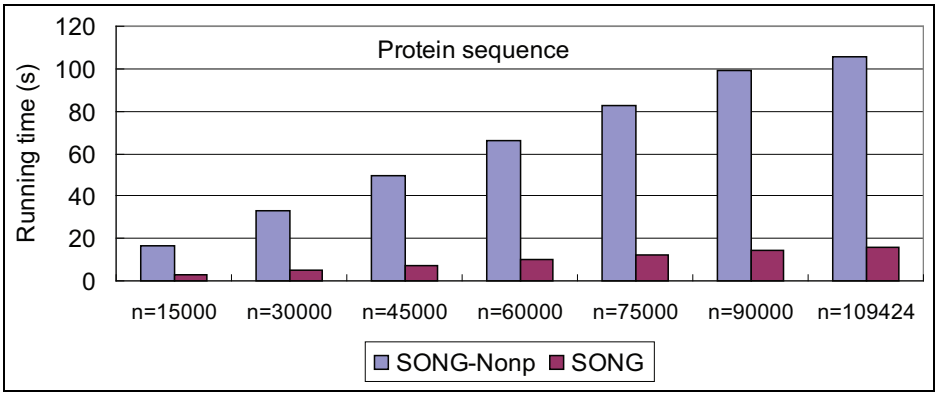

Figure 11. Comparison of algorithms running time on protein sequences.

m[3,15]g[3,39]e[3,35]f[0,10]g[0,15]t, a[0,5]t[0,7]c[0,9]g[0,11]g[0,10]v[0,10]g, a[0,10]v[0,10]g[0,10]a[0,15]c[0,15]t, c[0,15]t[0,15]e[0,15]a[0,15]a[0,8]g, t[0,10]v[0,10]g[0,15]t[0,15]e[0,5]c, r[0,10]g[0,10]y[0,15]a[0,15]a[0,9]e, t[0,12]g[0,12]y[0,10]g[0,15]t[0,8]c and a[0,12]t[0,12]g[0,12]g[0,12]y[0,10]g. In this section, *MinLen*, *MaxLen* and d are 15, 30 and 1, respectively. Therefore, 63 instances which are patterns from $Q1$ to $Q9$ and sequences from $T1$ to $T7$ are used to illustrate the scalability of the algorithms and the comparison of algorithms running time on each sequence is shown in Figure 11.

From Figure 11, it is easy to see that the running time of SONG-Nonp and SONG also grows linearly with n as well as in DNA sequences. Moreover, SONG is a more effective algorithm than SONG-Nonp. In solving all 63 instances, it costs SONG-Nonp and SONG about 453 and 68 s, respectively. Therefore, SONG is about 6.7 times faster than SONG-Nonp on protein sequences. We know that SONG is only about 1.7 times faster than SONG-Nonp on DNA sequences. The reason for this phenomenon is, a sequence of DNA is constituted by four kinds of characters, so the occurring possibility of each kind of character is 1/4, while the occurring possibility is 1/20 in protein. Therefore, the possibility of $N_s(n_1^r, n_j^{i_j-1}, j)$ getting 0 is higher in protein than that in DNA. According to Theorem 1, more sub-Nettrees will be removed in dealing with protein sequences. Hence, SONG can get better performance in protein.

6. Conclusion

In this study, we define a new approximate pattern matching problem with gap constraints, which is more complex than exact pattern matching. Theoretical analysis and experiments over real biological data together prove that the ratio $M(P,S,d)/N(P,S,m)$ approximately follows the binomial distribution $B(m,1\text{-}q)$, where $M(P,S,d)$ and $N(P,S,m)$ are the numbers of the approximate occurrences whose distances to pattern P are d ($0 \leq d \leq m$), and no more than m, respectively, and q is the sum of the products of the probabilities of each character c_l in pattern P and sequence S. In addition, we propose a new Single-root Nettree concept and design an efficient complete algorithm SONG with which to solve the problem. The time and space complexities of SONG are $O((n\text{-}m)*m^2*W^2*d)$ and $O(m^2*W*d)$, respectively, where n, m, W and d are sequence length, pattern length, maximal pattern gap and the similarity constraint, respectively. Experimental results obtained using real biological data verify the efficiency and correctness of SONG.

In this study, we focus on an approximate pattern matching with Hamming distance. The definitions of the problem will be changed significantly when using other distance functions, such as edit distance, because the gap constraints are considered in this issue. For example, there are many patterns whose edit distances to pattern a[1,2]t[1,3]c are 1, such as pattern ?[x,y]a[1,2]t[1,3]c, a[1,2]?[x,y]t[1,3]c, a[1,2]t[x,y]?[1,3]c and a[1,2]t[1,3]c[x,y]?, where x and y can be any integers and '?' can be any character. Therefore, to tackle the approximate pattern matching with other distance functions, first of all, new definitions should be given to deal with the gap constraints. Now, SONG cannot deal with other distance functions. The next step in this study is to explore the approximate pattern matching with other distance functions.

Acknowledgements

We thank the anonymous referees for their useful suggestions to improve this work.

Funding

This research is supported by the National Program on Key Basic Research Project under Grant 2013CB035906, the National Natural Foundation of China under grants No. 61229301, the Natural Science Foundation of Hebei Province of China under grant Nos. F2013202138 and G2014202031, and the Youth Foundation of Education Commission of Hebei Province under grant No. QN2014192.

References

[1] Won JI, Park S, Yoon JH and Kim SW. An efficient approach for sequence matching in large DNA databases. *Journal of Information Science* 2006; 32: 88–104.
[2] Wu Y, Wang L, Ren J, Ding W and Wu X. Mining sequential patterns with periodic wildcard gaps. *Applied Intelligence* 2014; 41: 99–116.
[3] Hong S and Kim M. Effective pattern-driven concurrency bug detection for operating systems. *Journal of Systems and Software* 2013; 86: 377–388.
[4] Akbari I and Fathian M. A novel algorithm for ontology matching. *Journal of Information Science* 2010; 36: 324–334.
[5] Hlayel AA and Hnaif A. An algorithm to improve the performance of string matching. *Journal of Information Science* 2014; 40: 357–362.
[6] Navarro G and Raffinot M. Fast and simple character classes and bounded gaps pattern matching with applications to protein searching. *Journal of Computational Biology* 2003; 10: 903–923.
[7] Cole R, Gottlieb L and Lewenstein M. Dictionary matching and indexing with errors and don't cares. In: *Proceedings of the 36th ACM Symposium on the Theory of Computing*, 2004, pp. 91–100.
[8] Califf M and Mooney R. Bottom-up relational learning of pattern matching rules for information extraction. *Journal of Machine Learning Research* 2003; 4: 177–210.
[9] Haapasalo T, Silvasti P, Sippu S and Soisalon SE. Online dictionary matching with variable-length gaps. *Experimental Algorithms* 2011; 6630: 76–87.
[10] Rahman S, Iliopoulos C, Lee I, Mohamed M and Smyth W. Finding patterns with variable length gaps or don't cares. *Computing and Combinatorics* 2006; 4112: 146–155.
[11] Sippu S and Soisalon-Soininen E. Online matching of multiple regular patterns with gaps and character classes. *Language and Automata Theory and Applications* 2013; 7810: 523–534.
[12] Wu Y, Liu Y, Guo L and Wu X. Subnettrees for strict pattern matching with general gaps and length constraints. *Journal of Software* 2013; 24: 915–932.
[13] Min F, Wu X and Lu Z. Pattern matching with independent wildcard gaps. In: *Proceedings of the 8th International Conference on Pervasive Intelligence and Computing*, 2009, pp. 194–199.
[14] Wu Y, Wu X, Min F and Li Y. A Nettree for pattern matching with flexible wildcard constraints. In: *Proceedings of the 2010 IEEE International Conference on Information Reuse and Integration*, 2010, pp. 109–114.
[15] Guo D, Hu X, Xie F and Wu X. Pattern matching with wildcards and gap-length constraints based on a centrality-degree graph. *Applied Intelligence* 2013; 39: 57–74.
[16] Zhang M, Kao B, Cheung D and Yip K. Mining periodic patterns with gap requirement from sequences. *ACM Transactions on Knowledge Discovery from Data* 2007; 1: 7-es.
[17] Ji X, Bailey J and Dong G. Mining minimal distinguishing subsequence patterns with gap constraints. *Knowledge and Information Systems* 2007; 11: 259–286.
[18] Li C, Yang Q, Wang J and Li M. Efficient mining of gap-constrained subsequences and its various applications. *ACM Transactions on Knowledge Discovery from Data* 2012; 6: 1–39.
[19] Zhu X and Wu X. Mining complex patterns across sequences with gap requirements. In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2007, pp. 2934–2940.
[20] He D, Wu X and Zhu X. SAIL-APPROX: An efficient on-line algorithm for approximate pattern matching with wildcards and length constraints. In: *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine*, 2007, pp. 151–158.
[21] Fischer MJ and Paterson M S. String matching and other products. In: *Proceedings of the 7th SIAM AMS Complexity of Computation*, 1974, pp. 113–125.
[22] Manber U and Baeza YR. An algorithm for string matching with a sequence of don't cares. *Information Processing Letters* 1991; 37: 133–136.
[23] Bille P, Gørtz IL, Vildhøj HW and Wind D. String matching with variable length gaps. *Theoretical Computer Science* 2012; 443: 25–34.
[24] Ding B, Lo D, Han J and Khoo SC. Efficient mining of closed repetitive gapped subsequences from a sequence database. In: *IEEE 25th International Conference on Data Engineering (ICDE), Shanghai, China*, 2009, pp. 1024–1035.

[25] Ferreira PG and Azevedo PJ. Protein sequence pattern mining with constraints. In: *European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Porto, Portugal*, 2005, pp. 96–107.
[26] Wu X, Zhu X, He Y and Arslan AN. PMBC: Pattern mining from biological sequences with wildcard constraints. *Computers in Biology and Medicine* 2013; 43: 481–492.
[27] Lam HT, Mörchen F and Fradkin D. Mining compressing sequential patterns. *Statistical Analysis and Data Mining* 2013; 7: 35–52.
[28] Wu Y, Wu X, Jiang H and Min F. A heuristic algorithm for MPMGOOC. *Chinese Journal of Computers* 2011, 34(8): 1452–1462.