IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON DATA-ENABLED INTELLIGENCE FOR DIGITAL HEALTH

Received July 4, 2019, accepted July 29, 2019, date of publication August 5, 2019, date of current version August 20, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2933044

Frequent Patterns Mining in DNA Sequence

NA DENG[1], XU CHEN[2], DESHENG LI[3], AND CAIQUAN XIONG[1]

[1]School of Computer, Hubei University of Technology, Wuhan 430068, China
[2]School of Information and Safety Engineering, Zhongnan University of Economics and Law, Wuhan 430073, China
[3]College of Mathematics, Physics and Information Engineering, Anhui University of Science and Technology, Huainan 233100, China

Corresponding author: Na Deng (iamdengna@163.com)

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFC1405403, in part by the National Natural Science Foundation of China under Grant 61075059, in part by the Philosophical and Social Sciences Research Project of Hubei Education Department under Grant 19Q054, in part by the Green Industry Technology Leding Project (Product Development Category) of the Hubei University of Technology under Grant CPYF2017008, in part by the Research Foundation for Advanced Talents of the Hubei University of Technology under Grant BSQD12131, in part by the Natural Science Foundation of Anhui Province under Grant 1708085MF161, in part by the Key Project of Natural Science Research of Universities in Anhui under Grant KJ2015A236, and in part by the Zhongnan University of Economics and Law under Grant 2722019JCT035 and Grant 2722019JCG074.

ABSTRACT As a common biological sequence, DNA sequences contain important information. The discovery of frequent patterns in DNA sequences can help to study the evolution, function and variation of genes. The findings are of great significance to genetic and mutation analysis, analysis of disease causes and treatment of diseases. Traditional methods of frequent pattern discovery need to scan DNA sequences multiple times. To overcome this shortcoming, this article proposes a new method to discover frequent patterns from DNA sequences. This method is based on a two-level nested hash table data structure and set operation. All frequent patterns and their positions in DNA sequences can be found by scanning DNA sequences only once. Experimental results show that this method can correctly recognize all the frequent patterns in DNA sequences and their locations. The method can also be applied to discover frequent patterns in RNA, protein or other biological sequences.

INDEX TERMS Big data, biological information, data mining, DNA sequence, frequent pattern, hash table.

I. INTRODUCTION

In the era of big data, with the rapid development of network and the extensive use of large capacity storage devices, a large amount of data has been accumulated in various fields. These data contain abundant knowledge and information. If we can discover the hidden information and make use of it, it will change the history of massive data but poor knowledge, which is important for the development of various fields.

There is a huge amount of data in the biological world, and these data contain important information. Discovering these hidden information can help speed up the process of biological research and is of great significance to the biological world. With the development of computer technology, the computer's powerful computing power has made it an important tool for biological research, thus a new subject named bioinformatics emerged [1]–[5].

In bioinformatics, biological sequence data is an important research object. Biological sequence data include three kinds of sequences: DNA sequence, RNA sequence and protein sequence. The molecular elements of these sequences can influence and determine the external shape and internal functions of organisms.

The replication of genes in the process of evolution produces many repetitive sequences, which can contribute to the production of new genes, and it is of great significance in genetic variation analysis. Mutations in repetitive sequences can lead to diseases [6]–[9], such as muscle atrophy, Williams syndrome, and thymus hypoplasia syndrome. The analysis and study of repeated sequences can be used to guide the research and interpretation of the functions of genes and non-gene sequences, which can help to understand the evolution and causes of gene mutation [10].

In order to explore the inherent law in DNA, a novel method is proposed in this paper. Based on a two-level nested hash table, frequent recurring patterns (i.e. frequent patterns) and their locations can be found through set operation. Compared with the traditional method, which needs to scan DNA sequence multiple times, our method scans DNA sequence only once, and at the same time, the positions of frequent patterns are recorded.

The associate editor coordinating the review of this manuscript and approving it for publication was Yongqiang Cheng.

II. RELATED WORK

Given the importance of DNA in life sciences, there are a lot of researches about DNA mining and analysis. Reference [11] made a survey on similarity analysis methods for DNA sequence. Reference [12] gave a novel model for DNA sequence similarity analysis based on graph theory. Adjacency matrix of directed graph is used to induce a representative vector for DNA sequence. To compute the similarity between DNA sequences, [13] introduced a novel method based on frequency patterns and entropy to construct representative vectors of DNA sequences

Frequent sequence discovery is a sub-problem in pattern recognition, there are many researches focusing on the frequent sequences discovery in DNA sequence. Traditional methods of discovering DNA frequent patterns usually made use of Apriori algorithm, that is, firstly collect frequent 1 patterns; then, generate candidate 2 patterns from frequent 1 patterns, and the like.

Since Apriori-based algorithm behaved poor performance while confronting long sequence, [14] proposed a algorithm named PrefixSpan to mines the complete set of patterns but greatly reduces the efforts of candidate subsequence generation. Reference [15] considered that PrefixSpan is not suitable for mining long frequent concatenate sequences; thus, it gave two algorithms MacosFSpan and MacosVSpan, to mine maximal frequent concatenate sequences. By constructing the spanning tree with a fixed length, [16] showed that its proposed method is much more efficient than MacosVSpan in terms of retrieval performance. By constructing a suffix tree, [17] proposed an efficient method to mine maximal contiguous frequent sub-sequences. To adapt to the era of big data, [18] proposed an efficient approach for mining maximal contiguous frequent patterns in large DNA sequence data using MapReduce framework on Hadoop platform.

In the traditional method of discovering DNA frequent patterns, the DNA sequence needs to be scanned for many times, the occurrence number of DNA fragments of different lengths is counted, and finally, all the DNA fragments which appear more than the threshold are seemed as frequent patterns. Usually, these methods usually aimed at finding frequent patterns, but ignored the positions information of the patterns. In this paper, we design a new method to discover frequent patterns just scanning the DNA sequence only once, and at the same time, all the position information is recorded.

III. CHARACTERISTICS

The two outstanding characteristics of the DNA sequence are as follows.

- DNA sequence has only four letters, i.e. A, C, G and T, representing four nucleotides constituting DNA. They are adenine, cytosine, guanine, thymine respectively.
- In DNA, the pairing between nucleotides is based on the complementary principle, that is, A is paired with T and G is paired with C.

IV. CONCEPTS OF DNA SEQUENCE

Definition 1 (DNA Sequence): DNA sequence is a series consisting of four letters (i.e. A, C, G and T). A DNA sequence can be represented as $S = \{s_0, s_1, s_2, \ldots\ldots, s_{n-1}\}$, in which, $s_i \in \{A, C, G, T\}(0 \leq i \leq n-1)$.

For example, {A, T, G, C, T, A, G} is a DNA sequence.

Definition 2 (The Length of DNA Sequence): For DNA sequence $S = \{s_0, s_1, s_2, \ldots\ldots, s_{n-1}\}$, n is called the length of this DNA sequence.

For example, the length of DNA sequence {A, T, G, C, T, A, G} is 7.

Definition 3: (DNA sub-Sequence): DNA sub-sequence is a fragment of DNA sequence.

For DNA sequence $S = \{s_0, s_1, s_2, \ldots\ldots, s_{n-1}\}$, if there is a fragment $S' = \{s_p, s_{p+1}, \ldots\ldots, s_{p+q}$ satisfying $p \geq 0$, $q \geq 0$ and $p+q \leq n-1$, then S' is called a DNA sub-sequence of S.

For example, {T, G, C, T} is a sub-sequence of DNA sequence {A, T, G, C, T, A, G}

Definition 4 (DNA Super-Sequence): Suppose there is a DNA sequence $S = \{s_0, s_1, s_2, \ldots\ldots, s_{n-1}\}$, S'_1 and S'_2 are two sub-sequences of S, and $S'_1 = \{s_p, s_{p+1}, \ldots\ldots, s_{p+q}\}$, $S'_2 = \{s_{p'}, s_{p'+1}, \ldots\ldots, s_{p'+q'}\}$. If $p' \leq p$ and $p+q \leq p'+q'$, then S'_2 is the super-sequence of S'_1.

For example, there is a DNA sequence $S = \{A, T, G, C, T, A, G\}$, $S'_1 = \{G, C, T\}$ and $S'_2 = \{T, G, C, T, A\}$ are two sub-sequences of S, then S'_2 is the super-sequence of S'_1.

Definition 5 (DNA Pattern): For DNA sequence S, its letter constitution is called the corresponding DNA pattern of S.

For example, "ATGCTAG" is the corresponding DNA pattern of DNA sequence {A, T, G, C, T, A, G}.

Definition 6 (The Length of DNA Pattern): The length of DNA pattern M is the number of letters in M, and denoted as length(M).

For example, the length of DNA pattern "ATGCTAG" is 7.

Definition 7 (DNA sub-Pattern): If S' is a sub-sequence of DNA sequence S, then the corresponding DNA pattern of S' is a sub-pattern of the corresponding DNA pattern of S.

For the example of Definition 3, pattern "GCT" is a sub-pattern of "ATGCTAG".

Definition 8 (DNA Super-Pattern): If S' is a super-sequence of DNA sequence S, then the corresponding DNA pattern of S' is a super-pattern of the corresponding DNA pattern of S.

For the example of Definition 4, "TGCTA" is a super-pattern of "TGC".

Definition 9 (DNA Frequent Pattern): Suppose *min_sup* is the minimum support degree threshold which is set manually. For DNA sequence S, if the corresponding DNA pattern of a sub-sequence S' appears in S not less than *min_sup*, then the corresponding DNA pattern of S' is a DNA frequent pattern in S.

For example, DNA sequence $S = \{A, T, C, G, A, T\}$, if *min_sup* is set to 2, pattern "AT" is a DNA frequent pattern in S.

Definition 10 (DNA Frequent k Pattern): The DNA frequent pattern with length k is called DNA frequent k pattern.

For the example in Definition 9, "AT" is a DNA frequent 2 pattern.

Theorem 1: The sub-pattern of a DNA frequent pattern is also frequent.

Proof of Theorem 1: Suppose there is a DNA sequence S, $S = \{s_0, s_1, s_2, \ldots\ldots, s_{n-1}\}$, S' is a sub-sequence of S, $S' = \{s_p, s_{p+1}, \ldots\ldots, s_{p+q}\}$, and the corresponding DNA pattern of S' is frequent in S.

Since the corresponding DNA pattern of S' is frequent, the occurrence number of $\{s_p, s_{p+1}, \ldots\ldots, s_{p+q}\}$ in S is not less than *min_sup*. Then, the occurrence number of any sub-pattern of S' in S is definitely not less than *min_sup*. Thus, the sub-pattern of a DNA frequent pattern is also frequent.

Theorem 2: The super-pattern of a DNA infrequent pattern is also infrequent.

Proof of Theorem 2: Suppose there is a DNA sequence S, $S = \{s_0, s_1, s_2, \ldots\ldots, s_{n-1}\}$, $S' = \{s_p, s_{p+1}, \ldots\ldots, s_{p+q}\}$ is a sub-sequence of S, and S'' is the super-sequence of S'. Since the corresponding DNA pattern of S' is infrequent, the occurrence number of $\{s_p, s_{p+1}, \ldots\ldots, s_{p+q}\}$ in S is less than *min_sup*. Thus, the occurrence number of S'' in S is definitely less than *min_sup*. Thus, the super-pattern of a DNA infrequent pattern is also infrequent.

Definition 11 (The Prefix of DNA Pattern): If the length of a DNA pattern M is more than 1, then the prefix of M is its substring removing the last letter. Any DNA pattern with length 1 has no prefix. The prefix of M is denoted as prefix(M).

For example, suppose a DNA pattern M = "ACTGA", then, prefix(M) = "ACTG".

Definition 12 (The Postfix of DNA Pattern): If the length of a DNA pattern M is more than 1, then the postfix of M is its substring removing the first letter. Any DNA pattern with length 1 has no postfix. The postfix of M is denoted as postfix(M).

For example, suppose a DNA pattern M = "ACTGA", then, postfix(M) = "CTGA".

Definition 13 (Joinable): Two patterns M_1 and M_2 are joinable if and only if either of the following two conditions are satisfied:

(1) When length(M_1) = length(M_2) = 1, M_1 and M_2 are joinable.

(2) When length(M_1) = length(M_2) > 1, M_1 and M_2 are joinable if and only if postfix(M_1) = prefix(M_2).

Definition 14 (String Concatenation): The string concatenation of two patterns M_1 and M_2 is denoted as M_1+M_2, which is the sequential concatenation of strings M_1 and M_2.

For example, pattern M_1 is "ATCGC", and pattern M_2 is "AGTC", then, the string concatenation of M_1 and M_2 is "ATCGCAGTC".

Definition 15 (The Join of Two Patterns): If two patterns M_1 and M_2 are joinable, the join of M_1 and M_2 is denoted as join(M_1, M_2), whose value depends on different conditions as follows:

(1) If length(M_1) = length(M_2) = 1, then, join(M_1, M_2) = $M_1 + M_2$, which is the string concatenation of M_1 and M_2. The example is shown in Fig. 1.

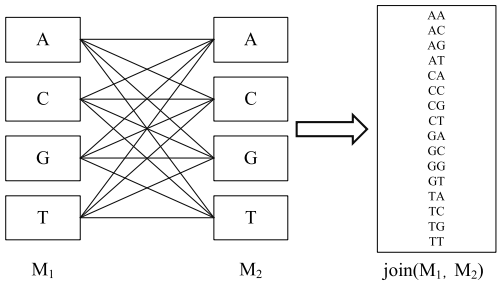


FIGURE 1. Join operation when length(M1) = length(M2) = 1.

(2) If length(M_1) = length(M_2) > 1, then, join(M_1, M_2) = M_1 + (M_2 − prefix(M_2)), which is the string concatenation of M_1 and the last letter of M_2. The example is shown in Fig. 2. The letters underlined represent the prefix or postfix of the patterns.

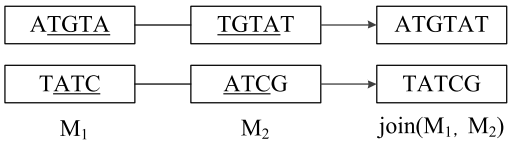


FIGURE 2. Join operation when length(M1) = length(M2) > 1.

It is worth noting that, when length(M_1) = length(M_2) = 1, M_1 and M_2 are joinable, the joined result is $M_1 + M_2$; in addition, M_2 and M_1 are joinable too, the joined result is $M_2 + M_1$.

Theorem 3: Any DNA frequent k + 1 pattern is joined by two DNA frequent k patterns.

Proof of Theorem 3: Suppose $m_1m_2 \ldots\ldots m_{k+1}$ is any DNA frequent k + 1 pattern. By Theorem 1, the sub-pattern of DNA frequent pattern is also frequent, so, the sub-pattern of $m_1m_2 \ldots\ldots m_{k+1}$ is also frequent. Thus, both $m_1m_2 \ldots\ldots m_k$ and $m_2m_3 \ldots\ldots m_{k+1}$ are frequent. In addition, $m_1m_2 \ldots\ldots m_{k+1}$ = join($m_1m_2 \ldots\ldots m_k$, $m_2m_3 \ldots\ldots m_{k+1}$), in other words, $m_1m_2 \ldots\ldots m_{k+1}$ is joined by $m_1m_2 \ldots\ldots m_k$ and $m_2m_3 \ldots\ldots m_{k+1}$.

Accordingly, any DNA frequent k + 1 pattern is joined by two DNA frequent k patterns.

Theorem 4: DNA candidate k + 1 pattern can be generated only through self join operation on the set of DNA frequent k patterns.

Proof of Theorem 4: It can be easily deduced from Theorem 3.

V. POSITION INFORMATION OF DNA

Definition 16: The positions of DNA pattern M in DNA sequence S is the index set of all occurrence of M in S, denoted as $\text{Positions}_{M,S}$.

The index of the first letter in a DNA sequence is stipulated as 0, and so forth.

For example, suppose a DNA sequence is {A, G, C, A, C, T, A, G, T, A, G, C}, the positions of DNA pattern "AGC" in this DNA sequence is {0, 9}.

Definition 17 (+n set): If there is a set $E = \{e_i\}$, satisfying $1 \le i \le |E|$, then, +n set of E is defined as $E^{+n} = \{e_i + n\}$, here, n is any natural number.

For example, $E = \{1, 4, 9\}$, then, $E^{+1} = \{2, 5, 10\}$, and $E^{+2} = \{3, 6, 11\}$.

Definition 18 (−n set): If there is a set $E = \{e_i\}$, satisfying $1 \le i \le |E|$, then -n set of E is defined as $E^{-n} = \{e_i - n | e_i - n \ge 0\}$, here, n is any natural number.

For example, $E = \{0, 1, 4, 9\}$, then, $E^{-1} = \{0, 3, 8\}$, and $E^{-2} = \{2, 7\}$.

Theorem 5: The positions of a DNA pattern M in DNA sequence S is two sets' intersection. These two sets are: the positions of M's prefix in S and -1 set of the positions of M's postfix in S. That is, $\text{Positions}_{M,S} = \text{Positions}_{\text{prefix(M), S}} \cap \text{Positions}^{-1}_{\text{postfix(M), S}}$.

Proof of Theorem 5 (We demonstrate): $\text{Positions}_{M,S} = \text{Positions}_{\text{prefix(M), S}} \cap \text{Positions}^{-1}_{\text{postfix(M), S}}$ from two aspects:

(1) We need to demonstrate Theorem 5.1:

Theorem 5.1: if $x \in \text{Positions}_{M,S}$, then $x \in \text{Positions}_{\text{prefix(M), S}} \cap \text{Positions}^{-1}_{\text{postfix(M), S}}$.

Proof of Theorem 5.1: Since $\text{Positions}_{M,S} \subseteq \text{Positions}_{\text{prefix(M), S}}$, so if $x \in \text{Positions}_{M,S}$, then $x \in \text{Positions}_{\text{prefix(M), S}}$.

If $x \in \text{Positions}_{M,S}$, then $x + 1 \in \text{Positions}_{\text{postfix(M), S}}$. Since $x \ge 0$, then $x + 1 \ge 1$, so $x \in \text{Positions}^{-1}_{\text{postfix(M), S}}$.

Thus, we get: $x \in \text{Positions}_{\text{prefix(M), S}} \cap \text{Positions}^{-1}_{\text{postfix(M), S}}$.

(2) We need to demonstrate Theorem 5.2:

Theorem 5.2: if $x \in \text{Positions}_{\text{prefix(M), S}} \cap \text{Positions}^{-1}_{\text{postfix(M), S}}$, then $x \in \text{Positions}_{M,S}$.

Proof of Theorem 5.2: If $x \in \text{Positions}_{\text{prefix(M), S}} \cap \text{Positions}^{-1}_{\text{postfix(M), S}}$, then $x \in \text{Positions}_{\text{prefix(M), S}}$ and $x \in \text{Positions}^{-1}_{\text{postfix(M), S}}$.

If $x \in \text{Positions}^{-1}_{\text{postfix(M), S}}$, then $x+1 \in \text{Positions}_{\text{postfix(M), S}}$.

If $x \in \text{Positions}_{\text{prefix(M), S}}$ and $x + 1 \in \text{Positions}_{\text{postfix(M), S}}$, then $x \in \text{Positions}_{M,S}$.

Consider Theorem 5.1 and 5.2 comprehensively, we can get Theorem 5.

Example of Theorem 5: Suppose there are a DNA sequence S "A, G, C, A, C, T, A, G, T, A, G, C", and a DNA pattern M "AGC", we have:

$$\begin{aligned}
\text{Positions}_{M,S} &= \{0, 9\} \\
\text{prefix (M)} &= \text{"AG"} \\
\text{postfix (M)} &= \text{"GC"} \\
\text{Positions}_{\text{prefix(M), S}} &= \{0, 6, 9\} \\
\text{Positions}_{\text{postfix(M), S}} &= \{1, 10\} \\
\text{Positions}^{-1}_{\text{postfix(M), S}} &= \{0, 9\} \\
\text{Positions}_{\text{prefix(M), S}} \cap \text{Positions}^{-1}_{\text{postfix(M), S}} &= \{0, 6, 9\} \cap \{0, 9\} \\
&= \{0, 9\}
\end{aligned}$$

VI. DATA STRUCTURE

In traditional frequent pattern mining methods, in order to discover the frequent patterns with different lengths, it is necessary to scan the sequence multiple times to count the occurrence number of frequency patterns with different lengths. According to Theorem 5, the positions of any DNA pattern in a sequence can be obtained by set operations using the positions of the pattern's prefix and postfix.

In order to reduce the number of sequence scanning, we use a nested hash table to save the positions of frequent patterns in the sequence, and its data structure is shown in Fig. 3.

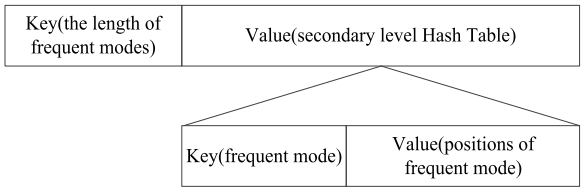


FIGURE 3. Nested two-level hash table storing DNA frequent patterns.

This data structure is a nested two-level hash table. In the first-level hash table, the Key stores the length of frequent patterns, and the Value is a nested hash table. The Key of this nested hash table stores frequent patterns, and the Value stores the locations of frequent patterns.

VII. SCANNING OF DNA SEQUENCE

In our method, we only need to scan the DNA sequence only once to find all the frequent patterns. In the scanning process, the location of each letter in DNA sequence is stored at the same time. The location information is stored in another simple hash table. The structure of this hash table is shown in Fig. 4. We call it DNA letter-position hash table.

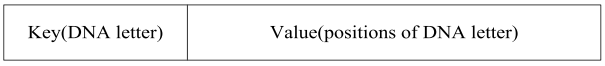


FIGURE 4. DNA letter-position hash table.

In this hash table, the Key stores DNA letters, i.e. A, G, C, T, and the Value stores all the positions of DNA letters in DNA sequence.

For example, suppose a DNA sequence is "A, G, C, A, C, T, A, G, T, A, G, C", after scanning the sequence once, the position information is stored in the following letter-position hash table, as shown in Fig. 5.

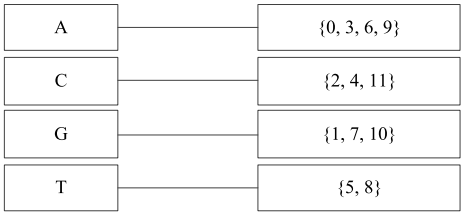


FIGURE 5. Example of DNA letter-position hash table.

VIII. DISCOVERING METHOD OF FREQUENT PATTERNS

A. WORKFLOW OF THE METHOD

The workflow of our method is shown in Fig. 6.

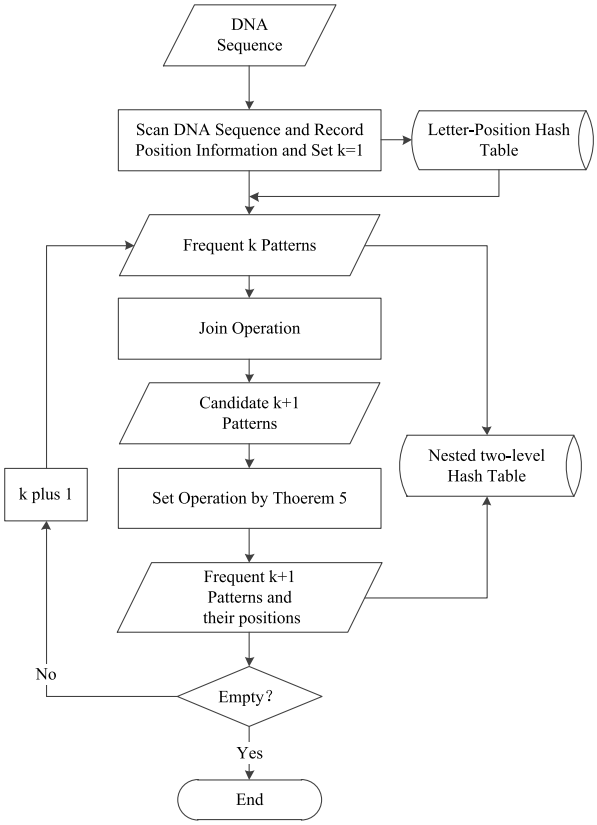


FIGURE 6. Workflow of the method.

B. ALGORITHM DESCRIPTION

1. Scan DNA sequence
   After scanning the DNA sequence S only once, a letter-position hash table is created. Suppose the minimum support threshold is *min_sup*, letters that appear not less than *min_sup* will be frequent 1 patterns.
2. Generate candidate 2 patterns from frequent 1 patterns
   Suppose the set of frequent 1 patterns is denoted as $\text{Freq} - 1 = \{t\}, t \in \{A, C, G, T\}$, satisfying $\text{Positions}_{t,S} \geq \text{min\_sup}$, then, the set of candidate 2 patterns is generated using: $\text{Candidate} - 2 = \text{join}(\text{Freq} - 1, \text{Freq} - 1) = \{t_1t_2\}$, satisfying $t_1, t_2 \in \{A,C,G,T\}$.
3. Using Theorem 5 to calculate the positions and occurrence times of candidate 2 patterns, those candidate 2 patterns with occurrence times not less than *min_sup* are reserved as frequent 2 patterns.
4. Start with $k = 2$, loop the following steps:
   4.1. The set of candidate $k + 1$ patterns is generated from the set of frequent k patterns, using: $\text{Candidate} - (k + 1) = \text{join}(\text{Freq-k}, \text{Freq-k}) = \{\text{join}(t_1, t_2)\}$ where $t_1, t_2 \in$ Freq-k.
   4.2. Using Theorem 5 to calculate the positions and occurrence times of candidate $k + 1$ patterns, those candidate $k + 1$ patterns with occurrence times not less than *min_sup*are reserved as frequent $k + 1$ patterns.
   4.3. If the set of frequent $k + 1$ pattern is not empty, k plus 1, return to Step 4; otherwise, the algorithm terminates.
5. Finally, the set of all frequent patterns is obtained as $\bigcup \text{Freq-k}, k \geq 1$.

C. EXAMPLE

We use a short toy DNA sequence fragment to describe the procedure of our method in details. Suppose there is a short DNA sequence fragment S ''ACTGCATGCTATGCATGCC'', and *min_sup* is set to 2. The working process of our method is as follows.

(1) The DNA sequence is scanned only once and each letter's position is stored in DNA letter-position hash table. For easy observation of DNA sequence, this toy DNA sequence and its position labels is shown in Table 1.

TABLE 1. Toy DNA sequence with position labels.

0	1	2	3	4	5	6	7	8	9
A	C	T	G	C	A	T	G	C	T
10	11	12	13	14	15	16	17	18	
A	T	G	C	A	T	G	C	C	

The letter-position hash table of this toy DNA sequence is shown in Table 2.

TABLE 2. The letter-position hash table of toy DNA sequence.

A	{0, 5, 10, 14}
G	{3, 7, 12, 16}
C	{1, 4, 8, 13, 17, 18}
T	{2, 6, 9, 11, 15}

(2) Since *min_sup* is 2, and all letters exist in DNA sequence more than twice, so all letters are kept as frequent 1 patterns.

(3) Generate candidate 2 patterns from frequent 1 patterns by join operation and calculate their positions by Theorem 5, as shown in Table 3.

(4) Those candidate 2 patterns with occurrence times not less than *min_sup* are reserved as frequent 2 patterns. These patterns and their positions are shown in Table 4, and they are added into nested two-level hash table.

(5) Generate candidate 3 patterns from frequent 2 patterns by join operation and calculate their positions by Theorem 5, as shown in Table 5.

(6) Those candidate 3 patterns with occurrence times not less than *min_sup* are reserved as frequent 3 patterns. These patterns and their positions are shown in Table 6, and they are added into nested two-level hash table.

TABLE 3. Candidate 2 patterns and their positions.

Candidate 2 patterns M	$Positions_{prefix(M),S}$	$Positions^{-1}_{postfix(M),S}$	$Positions_{M,S}=Positions_{prefix(M),S} \bigcap Positions^{-1}_{postfix(M),S}$
AA	{0, 5, 10, 14}	{4, 9, 13}	{0, 5, 10, 14}⋂{4, 9, 13}=∅
AG	{0, 5, 10, 14}	{2, 6, 11, 15}	{0, 5, 10, 14}⋂{2, 6, 11, 15}=∅
AC	{0, 5, 10, 14}	{0, 3, 7, 12, 16, 17}	{0, 5, 10, 14}⋂{0, 3, 7, 12, 16, 17}={0}
AT	{0, 5, 10, 14}	{1, 5, 8, 10, 14}	{0, 5, 10, 14}⋂{1, 5, 8, 10, 14}={5, 10, 14}
GA	{3, 7, 12, 16}	{4, 9, 13}	{3, 7, 12, 16}⋂{4, 9, 8, 13}=∅
GG	{3, 7, 12, 16}	{2, 6, 11, 15}	{3, 7, 12, 16}⋂{2, 6, 11, 15}=∅
GC	{3, 7, 12, 16}	{0, 3, 7, 12, 16, 17}	{3, 7, 12, 16}⋂{0, 3, 7, 12, 16, 17}={3, 7, 12, 16}
GT	{3, 7, 12, 16}	{1, 5, 8, 10, 14}	{3, 7, 12, 16}⋂{1, 5, 8, 10, 14}=∅
CA	{1, 4, 8, 13, 17, 18}	{4, 9, 13}	{1, 4, 8, 13, 17, 18}⋂{4, 9, 13}={4, 13}
CG	{1, 4, 8, 13, 17, 18}	{2, 6, 11, 15}	{1, 4, 8, 13, 17, 18}⋂{2, 6, 11, 15}=∅
CC	{1, 4, 8, 13, 17, 18}	{0, 3, 7, 12, 16, 17}	{1, 4, 8, 13, 17, 18}⋂{0, 3, 7, 12, 16, 17}={17}
CT	{1, 4, 8, 13, 17, 18}	{1, 5, 8, 10, 14}	{1, 4, 8, 13, 17, 18}⋂{1, 5, 8, 10, 14, 17}={1, 8}
TA	{2, 6, 9, 11, 15}	{4, 9, 13}	{2, 6, 9, 11, 15}⋂{4, 9, 13}={9}
TG	{2, 6, 9, 11, 15}	{2, 6, 11, 15}	{2, 6, 9, 11, 15}⋂{2, 6, 11, 15}={2, 6, 11, 15}
TC	{2, 6, 9, 11, 15}	{0, 3, 7, 12, 16, 17}	{2, 6, 9, 11, 15}⋂{0, 3, 7, 12, 16, 17}=∅
TT	{2, 6, 9, 11, 15}	{1, 5, 8, 10, 14}	{2, 6, 9, 11, 15}⋂{1, 5, 8, 10, 14}=∅

TABLE 4. Frequent 2 patterns and their positions.

Frequent 2 patterns	Positions of frequent 2 patterns
AT	{5, 10, 14}
GC	{3, 7, 12, 16}
CA	{4, 13}
CT	{1, 8}
TG	{2, 6, 11, 15}

(7) Generate candidate 4 patterns from frequent 3 patterns by join operation and calculate their positions by Theorem 5, as shown in Table 7.

(8) Those candidate 4 patterns with occurrence times not less than *min_sup* are reserved as frequent 4 patterns. These patterns and their positions are shown in Table 8, and they are added into nested two-level hash table.

(9) Generate candidate 5 patterns from frequent 4 patterns by join operation and calculate their positions by Theorem 5, as shown in Table 9.

TABLE 6. Frequent 3 patterns and their positions.

Frequent 3 patterns	Positions of frequent 3 patterns
ATG	{5, 10, 14}
GCA	{3, 12}
TGC	{2, 6, 11, 15}

(10) Since there is not any candidate 5 patterns occuring more than twice in S, thus the algorithm terminates. Here, we get all frequent patterns.

IX. THE TRADITIONAL METHOD

In order to verify the correctness of our method, it is necessary to compare our method with the traditional method to verify whether our method can find all the frequent patterns correctly. The traditional method uses sliding windows to scan DNA sequences several times and records the occurrence number of sub-patterns in different sizes of sliding windows. The workflow of the traditional method is shown in Fig. 7.

TABLE 5. Candidate 3 patterns and their positions.

Candidate 3 patterns M	$Positions_{prefix(M),S}$	$Positions^{-1}_{postfix(M),S}$	$Positions_{M,S}=Positions_{prefix(M),S} \bigcap Positions^{-1}_{postfix(M),S}$
ATG	{5, 10, 14}	{1, 5, 10, 14}	{5, 10, 14}⋂{1, 5, 10, 14}={5, 10, 14}
GCA	{3, 7, 12, 16}	{3, 12}	{3, 7, 12, 16}⋂{3, 12}={3, 12}
GCT	{3, 7, 12, 16}	{0, 7}	{3, 7, 12, 16}⋂{0, 7}={7}
CAT	{4, 13}	{4, 9, 13}	{4, 13}⋂{4, 9, 13}={4}
CTG	{1, 8}	{1, 5, 10, 14}	{1, 8}⋂{1, 5, 10, 14}={1}
TGC	{2, 6, 11, 15}	{2, 6, 11, 15}	{2, 6, 11, 15}⋂{2, 6, 11, 15}={2, 6, 11, 15}

TABLE 7. Candidate 4 patterns and their positions.

Candidate 4 patterns M	$Positions_{prefix(M),S}$	$Positions^{-1}_{postfix(M),S}$	$Positions_{M,S}=Positions_{prefix(M),S} \bigcap Positions^{-1}_{postfix(M),S}$
ATGC	{5, 10, 14}	{1, 5, 10, 14}	$\{5, 10, 14\} \bigcap \{1, 5, 10, 14\}=\{5, 10, 14\}$
TGCA	{2, 6, 11, 15}	{2, 11}	$\{2, 6, 11, 15\} \bigcap \{2, 11\}=\{2, 11\}$

TABLE 8. Frequent 4 patterns and their positions.

Frequent 4 patterns	Positions of frequent 4 patterns
ATGC	{5, 10, 14}
TGCA	{2, 11}

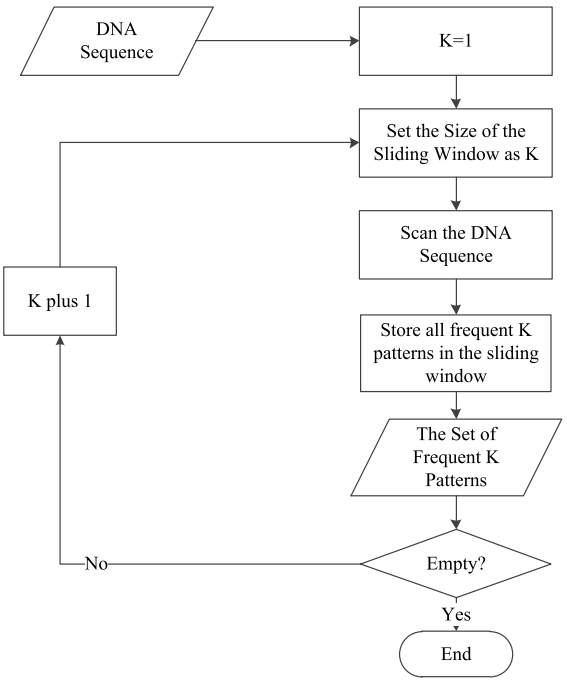


FIGURE 7. Workflow of the traditional method.

The steps of the traditional method are as follows:

1. Start with k = 1, loop the following steps:
   1.1. Set the size of the sliding window to k, scan the DNA sequence from front to back, record the occurrence number of patterns dropped in the sliding window gliding on the DNA sequence, and store the frequent k patterns with occurrence times not less than *min_sup* into the set denoted as Freq-k.
   1.2. If the set of frequent k pattern is not empty, k plus 1; otherwise, the algorithm terminates.
2. Finally, the set of all frequent patterns is obtained by $\bigcup$Freq-k,k $\geq$ 1.

A sketch map of sliding window with size 3 gliding on a DNA sequence fragment is shown in Fig. 8.

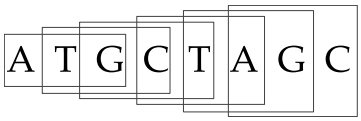


FIGURE 8. A sketch map of sliding window with size 3.

TABLE 10. Source code of generating DNA sequence with specific length randomly.

```
/**create a DNA string with specific length randomly**/
public static String  createDNA(int length)
{
    String[] DNA_elements= {"A","G","C","T"};
    String result="";
    Random random = new Random();
        for(int i=0;i<length;i++)
        {
            int element_ID=random.nextInt(4);
            String element=DNA_elements[element_ID];
            result+=element;
        }
        return result;
}
```

X. EXPERIMENTS

A. DATA SOURCE

Since DNA sequences are composed of four letters A, C, G, T, we use a random method to generate virtual DNA sequences in order to verify our method on DNA sequences with different lengths. In other words, the length of DNA sequence is set artificially, a random letter is selected from four letters at a time, and finally a DNA sequence that meets the length requirement is generated.

The source code of generating DNA sequence with specific length randomly in Java is shown in Table 10.

TABLE 9. Candidate 5 patterns and their positions.

Candidate 5 patterns M	$Positions_{prefix(M),S}$	$Positions^{-1}_{postfix(M),S}$	$Positions_{M,S}=Positions_{prefix(M),S} \bigcap Positions^{-1}_{postfix(M),S}$
ATGCA	{5, 10, 14}	{1, 10}	$\{5, 10, 14\} \bigcap \{1, 10\}=\{10\}$

TABLE 11. Experiment results.

Length	DNA sequence	min_sup	Frequent DNA patterns and their positions using two methods
100	CGGGGACGG TTTTCG TGTGTGGAC AGGCCACAT GTGCCCCGC GAAAATCTG ACGAAATCG GCAACAGGG CTATGGATC CCGCCCAGA CTTAGCAAA GCAA	8	2={CC=[27, 36, 37, 38, 77, 78, 81, 82], CA=[23, 28, 30, 61, 64, 83, 92, 97], GG=[1, 2, 3, 7, 20, 25, 59, 66, 67, 73], CG=[0, 6, 13, 39, 41, 52, 58, 79], GC=[26, 35, 40, 60, 68, 80, 91, 96], AA=[43, 44, 45, 54, 55, 62, 93, 94, 98]}, 1={A=[5, 22, 24, 29, 31, 43, 44, 45, 46, 51, 54, 55, 56, 62, 63, 65, 71, 75, 84, 86, 90, 93, 94, 95, 98, 99], T=[9, 10, 11, 12, 15, 17, 19, 32, 34, 47, 49, 57, 70, 72, 76, 88, 89], G=[1, 2, 3, 4, 7, 8, 14, 16, 18, 20, 21, 25, 26, 33, 35, 40, 42, 50, 53, 59, 60, 66, 67, 68, 73, 74, 80, 85, 91, 96], C=[0, 6, 13, 23, 27, 28, 30, 36, 37, 38, 39, 41, 48, 52, 58, 61, 64, 69, 77, 78, 79, 81, 82, 83, 87, 92, 97]}}
		5	{2={GG=[1, 2, 3, 7, 20, 25, 59, 66, 67, 73], AG=[24, 65, 84, 90, 95], GC=[26, 35, 40, 60, 68, 80, 91, 96], GA=[4, 21, 42, 50, 53, 74, 85], AC=[5, 22, 29, 51, 63, 86], AA=[43, 44, 45, 54, 55, 62, 93, 94, 98], CG=[0, 6, 13, 39, 41, 52, 58, 79], GT=[8, 14, 16, 18, 33], TG=[15, 17, 19, 32, 34, 49, 72], AT=[31, 46, 56, 71, 75], CC=[27, 36, 37, 38, 77, 78, 81, 82], CA=[23, 28, 30, 61, 64, 83, 92, 97]}, 1={A=[5, 22, 24, 29, 31, 43, 44, 45, 46, 51, 54, 55, 56, 62, 63, 65, 71, 75, 84, 86, 90, 93, 94, 95, 98, 99], T=[9, 10, 11, 12, 15, 17, 19, 32, 34, 47, 49, 57, 70, 72, 76, 88, 89], G=[1, 2, 3, 4, 7, 8, 14, 16, 18, 20, 21, 25, 26, 33, 35, 40, 42, 50, 53, 59, 60, 66, 67, 68, 73, 74, 80, 85, 91, 96], C=[0, 6, 13, 23, 27, 28, 30, 36, 37, 38, 39, 41, 48, 52, 58, 61, 64, 69, 77, 78, 79, 81, 82, 83, 87, 92, 97]}}
200	ATCTGACGA GGAGACGGG TGGTGGTGA GCAGGACAC TCATTAGGT GACCGCCAA CGCGAGTAC GGATTAACA TTGCACCGG CCACGAGCG GTCTT TAACGTTCA CGAGGAAGA TCAATAGCG GTTGGACCC GCATAGTTT AACACCTAC CAAGGACTT TT CATCTTGCA TGCAGTGTT ACGGGTAAC TATCTAAAT GCCT	10	{2={TT=[39, 66, 72, 93, 94, 100, 123, 137, 138, 156, 157, 158, 164, 176], GG=[9, 15, 16, 19, 22, 30, 42, 63, 79, 89, 107, 121, 125, 152, 180, 181], AG=[8, 11, 26, 29, 41, 58, 86, 106, 110, 118, 135, 151, 172], GC=[27, 49, 55, 74, 80, 87, 119, 131, 166, 170, 196], GA=[4, 7, 10, 12, 25, 31, 45, 57, 64, 85, 105, 108, 111, 126, 153], AC=[5, 13, 32, 34, 46, 53, 61, 69, 76, 83, 97, 103, 127, 141, 143, 147, 154, 178, 185], AA=[52, 68, 96, 109, 115, 140, 150, 184, 192, 193], CG=[6, 14, 48, 54, 56, 62, 78, 84, 88, 98, 104, 120, 130, 179], GT=[17, 20, 23, 43, 59, 90, 99, 122, 136, 173, 175, 182], TG=[3, 18, 21, 24, 44, 73, 124, 165, 169, 174, 195], AT=[0, 38, 65, 71, 112, 116, 133, 161, 168, 188, 194], CA=[28, 33, 37, 51, 70, 75, 82, 102, 114, 132, 142, 149, 160, 167, 171], TA=[40, 60, 67, 95, 117, 134, 139, 146, 177, 183, 187, 191]}, 1={A=[0, 5, 8, 11, 13, 26, 29, 32, 34, 38, 41, 46, 52, 53, 58, 61, 65, 68, 69, 71, 76, 83, 86, 96, 97, 103, 106, 109, 110, 112, 115, 116, 118, 127, 133, 135, 140, 141, 143, 147, 150, 151, 154, 161, 168, 172, 178, 184, 185, 188, 192, 193, 194], T=[1, 3, 18, 21, 24, 36, 39, 40, 44, 60, 66, 67, 72, 73, 91, 93, 94, 95, 100, 101, 113, 117, 123, 124, 134, 137, 138, 139, 146, 156, 157, 158, 159, 162, 164, 165, 169, 174, 176, 177, 183, 187, 189, 191, 195, 199], G=[4, 7, 9, 10, 12, 15, 16, 17, 19, 20, 22, 23, 25, 27, 30, 31, 42, 43, 45, 49, 55, 57, 59, 63, 64, 74, 79, 80, 85, 87, 89, 90, 99, 105, 107, 108, 111, 119, 121, 122, 125, 126, 131, 136, 152, 153, 166, 170, 173, 175, 180, 181, 182, 196], C=[2, 6, 14, 28, 33, 35, 37, 47, 48, 50, 51, 54, 56, 62, 70, 75, 77, 78, 81, 82, 84, 88, 92, 98, 102, 104, 114, 120, 128, 129, 130, 132, 142, 144, 145, 148, 149, 155, 160, 163, 167, 171, 179, 186, 190, 197, 198]}}
		6	{3={ACG=[5, 13, 53, 61, 83, 97, 103, 178], CGG=[14, 62, 78, 88, 120, 179], GGA=[9, 30, 63, 107, 125, 152], GGT=[16, 19, 22, 42, 89, 121, 181], GAG=[7, 10, 25, 57, 85, 105], GAC=[4, 12, 31, 45, 126, 153]}, 2={CT=[2, 35, 92, 145, 155, 163, 186, 190, 198], TT=[39, 66, 72, 93, 94, 100, 123, 137, 138, 156, 157, 158, 164, 176], GG=[9, 15, 16, 19, 22, 30, 42, 63, 79, 89, 107, 121, 125, 152, 180, 181], AG=[8, 11, 26, 29, 41, 58, 86, 106, 110, 118, 135, 151, 172], GC=[27, 49, 55, 74, 80, 87, 119, 131, 166, 170, 196], GA=[4, 7, 10, 12, 25, 31, 45, 57, 64, 85, 105, 108, 111, 126, 153], AC=[5, 13, 32, 34, 46, 53, 61, 69, 76, 83, 97, 103, 127, 141, 143, 147, 154, 178, 185], AA=[52, 68, 96, 109, 115, 140, 150, 184, 192, 193], CG=[6, 14, 48, 54, 56, 62, 78, 84, 88, 98, 104, 120, 130, 179], GT=[17, 20, 23, 43, 59, 90, 99, 122, 136, 173, 175, 182], TG=[3, 18, 21, 24, 44, 73, 124, 165, 169, 174, 195], AT=[0, 38, 65, 71, 112, 116, 133, 161, 168, 188, 194], CC=[47, 50, 77, 81, 128, 129, 144, 148, 197], CA=[28, 33, 37, 51, 70, 75, 82, 102, 114, 132, 142, 149, 160, 167, 171], TC=[1, 36, 91, 101, 113, 159, 162, 189], TA=[40, 60, 67, 95, 117, 134, 139, 146, 177, 183, 187, 191]}, 1={A=[0, 5, 8, 11, 13, 26, 29, 32, 34, 38, 41, 46, 52, 53, 58, 61, 65, 68, 69, 71, 76, 83, 86, 96, 97, 103, 106, 109, 110, 112, 115, 116, 118, 127, 133, 135, 140, 141, 143, 147, 150, 151, 154, 161, 168, 172, 178, 184, 185, 188, 192, 193, 194], T=[1, 3, 18, 21, 24, 36, 39, 40, 44, 60, 66, 67, 72, 73, 91, 93, 94, 95, 100, 101, 113, 117, 123, 124, 134, 137, 138, 139, 146, 156, 157, 158, 159, 162, 164, 165, 169, 174, 176, 177, 183, 187, 189, 191, 195, 199], G=[4, 7, 9, 10, 12, 15, 16, 17, 19, 20, 22, 23, 25, 27, 30, 31, 42, 43, 45, 49, 55, 57, 59, 63, 64, 74, 79, 80, 85, 87, 89, 90, 99, 105, 107, 108, 111, 119, 121, 122, 125, 126, 131, 136, 152, 153, 166, 170, 173, 175, 180, 181, 182, 196], C=[2, 6, 14, 28, 33, 35, 37, 47, 48, 50, 51, 54, 56, 62, 70, 75, 77, 78, 81, 82, 84, 88, 92, 98, 102, 104, 114, 120, 128, 129, 130, 132, 142, 144, 145, 148, 149, 155, 160, 163, 167, 171, 179, 186, 190, 197, 198]}}
300	GATATCACC CCGCCTGGA GCCTAGCAG TACCGGCCA TTACACTCA CCCCGAGAA TTCTGATAT GACACTAGT TTCCTTTGC ACAGCATGT ATGCCGCTT	12	{3={CAC=[5, 39, 43, 65, 80, 113, 130, 139, 194, 242, 244, 277, 280]}, 2={CT=[13, 20, 41, 56, 67, 75, 96, 106, 115, 126, 148, 152, 161, 171, 175, 190, 196, 202, 229, 246, 270, 274], TT=[36, 54, 71, 72, 76, 77, 97, 98, 120, 168, 176, 225, 226, 230, 231, 232, 236, 275], GG=[15, 31, 100, 111, 118, 143, 146, 173, 192, 266, 267, 288, 289, 293, 294], AG=[17, 22, 25, 50, 69, 83, 117, 142, 156, 186, 207, 211, 217, 234, 256, 261, 287, 292], GC=[11, 18, 23, 32, 79, 84, 92, 95, 101, 112, 122, 128, 144, 147, 154, 170, 174, 193, 251, 254, 297], GA=[0, 16, 49, 51, 58, 63, 135, 157, 165, 183, 208, 212, 218, 238, 248, 257, 262, 290], AC=[6, 28, 38, 40, 44, 64, 66, 81, 105, 114, 131, 136, 140, 150, 159, 184, 195, 201, 215, 241, 243, 245, 249, 264, 278, 281, 285], AA=[52, 158, 166, 200, 204, 205, 206, 209, 210, 219, 263, 291], CG=[10, 30, 48, 94, 123, 132, 145, 164, 250, 253, 265, 298], GT=[26, 70, 88, 119, 124, 133, 187, 198, 224, 235, 268, 272, 295], TG=[14, 57, 62, 78, 87, 91, 99, 110, 121, 127, 134, 153, 169, 172, 182, 191, 197, 223,

TABLE 11. *(Continued.)* **Experiment results.**

	TGGCATACT CATGGCACT AGGTTGCGT CTGCCACGT GACCCACAG GCGGCTACC TGCAGAACC TCCGAATTG CTGGCTTAT CATGACAGT CCTGGCACT GTAACTAAA AGAAAGATA CAGAATATG TTTCC TTTTAGTTG ATACACACT GACGCCGCA GATCAGAAC GGGTCTGTC		237, 247, 271, 296], AT=[1, 3, 35, 53, 59, 61, 86, 90, 103, 109, 167, 178, 181, 213, 220, 222, 239, 258, 283], CC=[7, 8, 9, 12, 19, 29, 33, 45, 46, 47, 74, 93, 129, 137, 138, 151, 160, 163, 189, 228, 252, 279], CA=[5, 24, 34, 39, 43, 65, 80, 82, 85, 102, 108, 113, 130, 139, 141, 155, 180, 185, 194, 216, 242, 244, 255, 260, 277, 280, 282, 286], TC=[4, 42, 55, 73, 107, 125, 162, 179, 188, 227, 259, 269, 273, 276], TA=[2, 21, 27, 37, 60, 68, 89, 104, 116, 149, 177, 199, 203, 214, 221, 233, 240, 284]}, 1={A=[1, 3, 6, 17, 22, 25, 28, 35, 38, 40, 44, 50, 52, 53, 59, 61, 64, 66, 69, 81, 83, 86, 90, 103, 105, 109, 114, 117, 131, 136, 140, 142, 150, 156, 158, 159, 166, 167, 178, 181, 184, 186, 195, 200, 201, 204, 205, 206, 207, 209, 210, 211, 213, 215, 217, 219, 220, 222, 234, 239, 241, 243, 245, 249, 256, 258, 261, 263, 264, 278, 281, 283, 285, 287, 291, 292], T=[2, 4, 14, 21, 27, 36, 37, 42, 54, 55, 57, 60, 62, 68, 71, 72, 73, 76, 77, 78, 87, 89, 91, 97, 98, 99, 104, 107, 110, 116, 120, 121, 125, 127, 134, 149, 153, 162, 168, 169, 172, 176, 177, 179, 182, 188, 191, 197, 199, 203, 214, 221, 223, 225, 226, 227, 230, 231, 232, 233, 236, 237, 240, 247, 259, 269, 271, 273, 275, 276, 284, 296], G=[0, 11, 15, 16, 18, 23, 26, 31, 32, 49, 51, 58, 63, 70, 79, 84, 88, 92, 95, 100, 101, 111, 112, 118, 119, 122, 124, 128, 133, 135, 143, 144, 146, 147, 154, 157, 165, 170, 173, 174, 183, 187, 192, 193, 198, 208, 212, 218, 224, 235, 238, 248, 251, 254, 257, 262, 266, 267, 268, 272, 288, 289, 290, 293, 294, 295, 297, 299], C=[5, 7, 8, 9, 10, 12, 13, 19, 20, 24, 29, 30, 33, 34, 39, 41, 43, 45, 46, 47, 48, 56, 65, 67, 74, 75, 80, 82, 85, 93, 94, 96, 102, 106, 108, 113, 115, 123, 126, 129, 130, 132, 137, 138, 139, 141, 145, 148, 151, 152, 155, 160, 161, 163, 164, 171, 175, 180, 185, 189, 190, 194, 196, 202, 216, 228, 229, 242, 244, 246, 250, 252, 253, 255, 260, 265, 270, 274, 277, 279, 280, 282, 286, 298]}}
	TTCACCACA TACAGGGAA GGGTGCG	8	{3={CAC=[5, 39, 43, 65, 80, 113, 130, 139, 194, 242, 244, 277, 280], CTG=[13, 56, 126, 152, 171, 190, 196, 246, 270], GCA=[23, 79, 84, 101, 112, 154, 193, 254], ACA=[38, 64, 81, 140, 184, 215, 241, 243, 281, 285], CAG=[24, 82, 141, 155, 185, 216, 255, 260, 286]}, 2={CT=[13, 20, 41, 56, 67, 75, 96, 106, 115, 126, 148, 152, 161, 171, 175, 190, 196, 202, 229, 246, 270, 274], TT=[36, 54, 71, 72, 76, 77, 97, 98, 120, 168, 176, 225, 226, 230, 231, 232, 236, 275], GG=[15, 31, 100, 111, 118, 143, 146, 173, 192, 266, 267, 288, 289, 293, 294], AG=[17, 22, 25, 50, 69, 83, 117, 142, 156, 186, 207, 211, 217, 234, 256, 261, 287, 292], GC=[11, 18, 23, 32, 79, 84, 92, 95, 101, 112, 122, 128, 144, 147, 154, 170, 174, 193, 251, 254, 297], GA=[0, 16, 49, 51, 58, 63, 135, 157, 165, 183, 208, 212, 218, 238, 248, 257, 262, 290], AC=[6, 28, 38, 40, 44, 64, 66, 81, 105, 114, 131, 136, 140, 150, 159, 184, 195, 201, 215, 241, 243, 245, 249, 264, 278, 281, 285], AA=[52, 158, 166, 200, 204, 205, 206, 209, 210, 219, 263, 291], CG=[10, 30, 48, 94, 123, 132, 145, 164, 250, 253, 265, 298], GT=[26, 70, 88, 119, 124, 133, 187, 198, 224, 235, 268, 272, 295], TG=[14, 57, 62, 78, 87, 91, 99, 110, 121, 127, 134, 153, 169, 172, 182, 191, 197, 223, 237, 247, 271, 296], AT=[1, 3, 35, 53, 59, 61, 86, 90, 103, 109, 167, 178, 181, 213, 220, 222, 239, 258, 283], CC=[7, 8, 9, 12, 19, 29, 33, 45, 46, 47, 74, 93, 129, 137, 138, 151, 160, 163, 189, 228, 252, 279], CA=[5, 24, 34, 39, 43, 65, 80, 82, 85, 102, 108, 113, 130, 139, 141, 155, 180, 185, 194, 216, 242, 244, 255, 260, 277, 280, 282, 286], TC=[4, 42, 55, 73, 107, 125, 162, 179, 188, 227, 259, 269, 273, 276], TA=[2, 21, 27, 37, 60, 68, 89, 104, 116, 149, 177, 199, 203, 214, 221, 233, 240, 284]}, 1={A=[1, 3, 6, 17, 22, 25, 28, 35, 38, 40, 44, 50, 52, 53, 59, 61, 64, 66, 69, 81, 83, 86, 90, 103, 105, 109, 114, 117, 131, 136, 140, 142, 150, 156, 158, 159, 166, 167, 178, 181, 184, 186, 195, 200, 201, 204, 205, 206, 207, 209, 210, 211, 213, 215, 217, 219, 220, 222, 234, 239, 241, 243, 245, 249, 256, 258, 261, 263, 264, 278, 281, 283, 285, 287, 291, 292], T=[2, 4, 14, 21, 27, 36, 37, 42, 54, 55, 57, 60, 62, 68, 71, 72, 73, 76, 77, 78, 87, 89, 91, 97, 98, 99, 104, 107, 110, 116, 120, 121, 125, 127, 134, 149, 153, 162, 168, 169, 172, 176, 177, 179, 182, 188, 191, 197, 199, 203, 214, 221, 223, 225, 226, 227, 230, 231, 232, 233, 236, 237, 240, 247, 259, 269, 271, 273, 275, 276, 284, 296], G=[0, 11, 15, 16, 18, 23, 26, 31, 32, 49, 51, 58, 63, 70, 79, 84, 88, 92, 95, 100, 101, 111, 112, 118, 119, 122, 124, 128, 133, 135, 143, 144, 146, 147, 154, 157, 165, 170, 173, 174, 183, 187, 192, 193, 198, 208, 212, 218, 224, 235, 238, 248, 251, 254, 257, 262, 266, 267, 268, 272, 288, 289, 290, 293, 294, 295, 297, 299], C=[5, 7, 8, 9, 10, 12, 13, 19, 20, 24, 29, 30, 33, 34, 39, 41, 43, 45, 46, 47, 48, 56, 65, 67, 74, 75, 80, 82, 85, 93, 94, 96, 102, 106, 108, 113, 115, 123, 126, 129, 130, 132, 137, 138, 139, 141, 145, 148, 151, 152, 155, 160, 161, 163, 164, 171, 175, 180, 185, 189, 190, 194, 196, 202, 216, 228, 229, 242, 244, 246, 250, 252, 253, 255, 260, 265, 270, 274, 277, 279, 280, 282, 286, 298]}}

B. ACCURACY VERIFICATION

From Theorem 5 we know that, the positions of patterns can be obtained through set operation; thus, all the frequent patterns and their positions will be found and recorded using the method proposed in Section VIII.

In addition, the correctness of the method can also be verified on the randomly generated DNA sequences of different lengths. The results show that the frequent patterns obtained by the two methods are totally the same, which proves the accuracy of the method, as seen in Table 11. Besides that, we can also see that, under different lengths, our method can find all the frequent patterns and their positions in DNA sequence.

In Table 11, frequent patterns and their positions are stored in the form of:

{the length of frequent pattern = {frequent pattern = [positions]}}

Each "{}" pair represents a single-level hash table. The positions of frequent patterns in DNA sequence are stored in "[ ]". In DNA sequence, the position of letters starts from 0.

TABLE 12. Running time of our method and the traditional method on DNA sequences with different lengths.

Length of randomly generated DNA sequence	10000	20000	30000	40000	50000	60000	70000	80000	90000
Running time of our method (millisecond)	375	997	2108	3479	5253	6617	8557	11967	17108
Running time of the traditional method (millisecond)	63	140	174	157	172	229	297	408	561

TABLE 13. Average running time of our method and the traditional method on DNA sequences with different lengths (the number of iterations is 10).

Length of randomly generated DNA sequence	10000	20000	30000	40000	50000	60000
Average running time of our method (millisecond)	230	760	1723	3409	4908	7139
Average running time of the traditional method (millisecond)	16	32	50	70	90	122

C. COMPARISION

Our method is based on two hash tables. Scanning the DNA sequence only once, each DNA letter and all its locations are stored in the letter-position hash table. After that, generate candidate k + 1 patterns from frequent k patterns, only through join operation. By using Theorem 5, perform set operations on the set of the locations of frequent patterns stored in the secondary hash table, then, the occurrence number of candidate k + 1 patterns in DNA sequences is calculated.

In the process of discovering all the frequent patterns in DNA sequences, our method scans the DNA sequences only once. However, the number of times the traditional method scans the DNA sequences depends on the length of the maximum frequent patterns; that is, the value of k when the traditional method stops.

D. RUNNING TIME

To compare the running time of our method and the traditional method, several DNA sequences with different lengths are created randomly. These two methods run on the same DNA sequences for comparison (*min_sup*=7). Table 12 shows their running time data.

Since DNA sequences generated at each time are different, for specific DNA length, multiple DNA sequences were randomly generated. The running time of these two methods was recorded for different DNA sequences, and the average running time of each method was calculated, as shown in Table 13.

E. ANALYSIS

Compared with the traditional method, our method consumes more time, this is because in our method, after scanning DNA sequence, all characters and their positions need to be stored in the hash table; and then, in each subsequent iteration, a large number of set operations are needed to perform through Theorem 5.

The apparent advantage of our method is that, the traditional method has to scan the DNA sequence many times to discover the frequent patterns. However, in our method, all position information of each element has been recorded in the only once scan, and frequent patterns and their position information can be calculated through set operation.

XI. CONCLUSIONS

The traditional method of discovering frequent patterns from DNA sequence needs to scan the DNA sequence several times, record the occurrence times of DNA sub-patterns with different lengths, and then take those whose occurrence times not less than the minimum support threshold as frequent patterns. In order to avoid multiple scanning of DNA sequence, two data structures: a simple one-level hash table and a two-level nested hash table, are introduced in. The letters and their positions in DNA sequence are stored in a simple letter-position hash table by scanning the DNA sequence only once. On the basis of this hash table, utilizing the set operations of location information, candidate k + 1 patterns are generated from frequent k patterns, and their location information is calculated. Finally, all DNA frequent patterns are obtained. The experiments show that the proposed method can get exactly the same results as the traditional method, and its correctness is verified. This method can be extended to discover frequent patterns in RNA sequences or protein sequences.

REFERENCES

[1] C. P. E. Agbachi, "Pathways in bioinformatics: A window in computer science," *Int. J. Comput. Trends Technol.*, vol. 49, no. 2, pp. 83–90, 2017.

[2] P. Tomasz, W. Szymon, and B. Jacek, "Computer representations of bioinformatics models," *Current Bioinform.*, vol. 11, nol. 5, pp. 551–560, 2016.

[3] M. Sardaraz, M. Tahir, and A. A. Ikram, "Advances in high throughput DNA sequence data compression," *J. Bioinf. Comput. Biol.*, vol. 14, no. 3, p. 18, 2016.

[4] S. X. Ge, "Exploratory bioinformatics investigation reveals importance of 'junk' DNA in early embryo development," *BMC Genomics*, vol. 18, no. 1, p. 200, 2017.

[5] S. Chen, M. Liu, and Y. Zhou, "Bioinformatics analysis for cell-free tumor DNA sequencing data," in *Computational Systems Biology*. New York, NY, USA: Humana Press, 2018, pp. 67–95.

[6] J. Zhang and K. Huang, "Pan-cancer analysis of frequent DNA co-methylation patterns reveals consistent epigenetic landscape changes in multiple cancers," *BMC Genomics*, vol. 18, p. 1045, Jan. 2017.

[7] R. Maloney, M. Budiman, Y. Korshunova, J. Monte, B. Bacher, N. Lakey, J. Leon, R. Martienssen, J. Jeddeloh, D. Herbert, J. McPherson, and J. Ordway, "Tissue-specific DNA methylation patterns are frequent targets of epigenetic change in multiple cancer types," *Cancer Res.*, vol. 68, p. 256, May 2008.

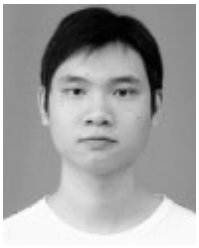
[8] R. Alves, D. S. Rodriguez-Baena, and J. S. Aguilar-Ruiz, "Gene association analysis: A survey of frequent pattern mining from gene expression data," *Briefings Bioinf.*, vol. 11, no. 2, pp. 210–224, 2010.

[9] M. Meggendorfer, T. Haferlach, T. Alpermann, S. Jeromin, C. Haferlach, W. Kern, and S. Schnittger, "Specific molecular mutation patterns delineate chronic neutrophilic leukemia, atypical chronic myeloid leukemia, and chronic myelomonocytic leukemia," *Haematologica*, vol. 99, no. 12, pp. e244–e246, 2014.

[10] S. Bai, "The maximum frequent pattern mining in DNA sequence," Ph.D. dissertation, School Inf. Eng., Nanchang Univ., Nanchang, China, 2010.
[11] X. Jin, Q. Jiang, Y. Chen, S.-J. Lee, R. Nie, S. Yao, D. Zhou, and K. He, "Similarity/dissimilarity calculation methods of DNA sequences: A survey," *J. Mol. Graph. Model.*, vol. 76, pp. 342–355, Sep. 2017.
[12] X. Qi, Q. Wu, Y. Zhang, E. Fuller, and C.-Q. Zhang, "A novel model for DNA sequence similarity analysis based on graph theory," *Evol. Bioinf.*, vol. 7, pp. 149–158, Oct. 2011.
[13] X. Xie, J. Guan, and S. Zhou, "Similarity evaluation of DNA sequences based on frequent patterns and entropy," *BMC Genomics*, vol. 16, no. 3, p. S5, 2015.
[14] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "PrefixSpan,: Mining sequential patterns efficiently by prefix-projected pattern growth," in *Proc. 17th Int. Conf. Data Eng.* Berlin, Germany, Apr. 2001, pp. 215–224.
[15] J. Pan, P. Wang, W. Wang, B. Shi, and G. Yang, "Efficient algorithms for mining maximal frequent concatenate sequences in biological datasets," in *Proc. 5th Int. Conf. Comput. Inf. Technol.*, Shanghai, China, Sep. 2005, pp. 98–104.
[16] T. H. Kang, J. S. Yoo, and H. Y. Kim, "Mining frequent contiguous sequence patterns in biological sequences," in *Proc. 7th Int. Symp. Bioinf. BioEng.* Boston, MA, USA, Oct. 2007, pp. 723–728.
[17] M. R. Karim, M. Rashid, B.-S. Jeong, and H.-J. Choi, "An efficient approach to mining maximal contiguous frequent patterns from large dna sequence databases," *Genomics Informat.*, vol. 10, no. 1, pp. 51–57, 2012.
[18] M. R. Karim, A. Hossain, M. Rashid, B.-S. Jeong, and H.-J. Choi, "A MapReduce framework for mining maximal contiguous frequent patterns in large DNA sequence datasets," *IETE Tech. Rev.*, vol. 29, no. 2, pp. 162–168, 2012.

NA DENG was born in Wuhan, China, in 1985. She received the M.S. and Ph.D. degrees in computer science and technology from the Beijing University of Post and Telecommunications, Beijing, China, in 2008 and 2011, respectively.

Since 2012, she has been a Lecturer with Hubei University of Technology, China. She is the author of one scholarly monograph, more than 40 articles, two patents, and eight software copyrights. Her research interests include web services, data mining, and artificial intelligence.



XU CHEN was born in Wuhan, China, in 1984. He received the M.S. and Ph.D. degrees in computer software and theory from Wuhan University, Wuhan, China, in 2009 and 2013, respectively.

Since 2013, he has been a Lecturer with Zhongnan University of Economics and Law, China. He is the author of more than 30 articles. His research interests include text mining and machine learning.

DESHENG LI was born in Yichang, China, in 1979. He received the M.S. and Ph.D. degrees in computer science and technology from the Beijing University of Post and Telecommunications, Beijing, China, in 2008 and 2011, respectively.

Since 2016, he has been a Professor with Anhui University of Science and Technology, China. He is the author of more than 50 articles and three patents. His research interests include swarm intelligence and artificial neural networks.

CAIQUAN XIONG was born in Ezhou, China, in 1966. He received the M.S. and Ph.D. degrees in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2001 and 2008, respectively.

He is currently a Professor with Hubei University of Technology, China. He is the author of one scholarly monograph, more than 60 articles, and one patent. His research interests include pattern recognition and artificial intelligence.

Dr. Xiong received one Third Prize of the Wuhan Science and Technology Progress, two Second Prizes of Hubei Teaching Achievement, and one First Prize and one Second Prize of the Hubei University of Technology.