AMSE JOURNALS-2016-Series: Advances A; Vol. 53; N°2; pp 34-47
Submitted July 2016; Revised Oct. 15, 2016, Accepted Dec. 10, 2016

Indexing DNA Sequence k-mer based on LNDM Algorithm

Pinchao Meng, Weishi Yin, Zhixia Jiang

Department of Applied Mathematics, College of Science ,Changchun university of Science and Technology
Changchun,Changchun Weixing Road No. 7089, (mengpc@cust.edu.cn)

Abstract

Quickly and accurately locate the target DNA sequence in the DNA library position, plays an important role in the development of bioinformatics. The thesis established an indexing model that is based on LNDM algorithm, designed two simplified linear structured automata, least suffix automaton and finite state automaton, because of their superior performance in string matching, located k-mer in the given database and analyzed the property of the algorithm. The results show that the index model of LNDM algorithm can quickly locate the positions of target short DNA strands in the DNA bank.

Key words

LNDM algorithm, automaton, indexing model, string matching

1. Introduction

Sequence data index is an important means of DNA data analysis. Locating the target DNA strand in a large gene data bank can enhance the development of bioscience. Thus, it is of great significance on biology to establish an accurate indexing model[1-6].

Assuming that a DNA sequence is given, which contains four letters ATCG,for example, S ="CTGTACTGTAT",and that an integer k is given,a short string with k letters starting from the first letter in S is called k-mer (for example, if k= 5,the string would be CTGTA). Then another string starting from the second letter in S would be another k-mer (for example, if k= 5,the string would be TGTAC). Different strings are acquired through starting with a different letter till the

end of S, which can form a set that includes all k-mers. For example, for the sequence S,all 5-mers are:

{CTGTA,TGTAC,GTACT,TACTG,ACTGT,TGTAT}

Usually, there needs to be an indexing model for these k-mers so that they can be quickly accessed. For example, for 5-mer, when CTGTA needs to be accessed, this kind of indexing method can locate it in the DNA sequence S.

This thesis will first provide an indexing method for a given k that can help access the serial number and location in the sequence for any k-mer, and then analyze the complexity of establishing the index and the storage needed.

2. Data Indexing of DNA Sequence

2.1 Establishing inverse least suffix automaton

Suffix automaton is an alphabetic tree. The string it records are all the suffixes of a given string s. The function of finite state automaton is to recognize strings. If an automaton A can recognize string S, then A(S)=True, otherwise A(S)=False. Hence, SAM(x) = True, for a given string S, only when x is the suffix of S. For example, for the string daabbabd, an alphabetic tree as shown in Figure 1 can be established.

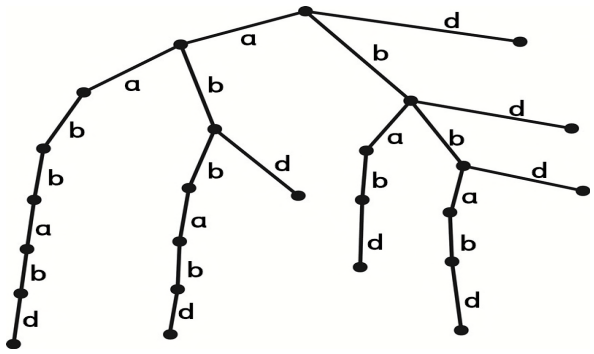

Fig.1. Tree automata

Since the state number of DNA sequence is usually large, the thesis proposes to transform the above complicated alphabetic tree into a simple linear structure using suffix automaton, which will not influence the result of indexing. The above alphabetic tree has plenty of nodes, however, there is only one son for the majority of nodes. In addition, they share a lot in common. Hence, given the large common part they share, the space can be compressed. To be specific, the edge-node that borders the son can be deleted (as well as the son and its offspring). Subsequently, it can be connected to other sub-trees, as a result of which, the common part can be made full use of, saving space. In addition, for suffix automaton, a certain node can be the son of multiple

nodes, which can ensure that there will be no repetition in the strings traversed from the suffix automaton, and that the strings are all the sub-strings of S.

In the suffix automaton, the information each node stores include:

A son: the corresponding location of the legal sub-string in suffix automaton generated by adding the node's corresponding sub-string and a certain character. If the pointer does not exist, it means that such a sub-string does not exist (or it is not a sub-string of s).

B Pre: it is important to note that it is not accessing its father node (because a certain node can be the son of multiple nodes), but accessing the last node that can receive suffix.

C step: the maximum steps that are needed from the root node to the particular node.

To more conveniently elaborate the establishing of indexing model, three points on the properties of suffix automaton are mentioned:

The strings made of characters on each path from the root node to node p are all sub-strings of the string t.

Due to the first property, if the node p can receive new suffixes, then the strings made of characters on each path from the root node to node p must all be suffixes of the string t.

If node p can receive new suffixes, then the node the pre of p points to can also receive suffixes. If not, then node the pre of p points to cannot, either.

The general method of simplifying suffix automaton:

Assuming that the suffix automaton t of a certain prefix of string s is already established, by adding a character x, the suffix automaton of another prefix tx of s can be acquired. Hence, by adding a character each time, and finishing all the characters of s, the suffix automaton of s can be acquired. Thus, the process of establishing suffix automaton is online, which means information of s can be accessed at any time, and a new character can be added anytime to the end of string s to make a new string.

First, establish a node np that stores the particular character x; find the last node to be established (because it must have the second property); then follow the pointer of pre until the node that has x as the son. Assume that it gets to the node p. If p does not have x as the son, it must be able to receive new characters. Give the son of node p a value np (now p has received the suffix character x and cannot receive new ones). In this way, nodes that have x as their sons can be processed. Assuming that the son of p is q, there are only two situations:

step[q]=step[p]+1.

To make sure that the number of nodes of suffix automaton is as small as possible and that they share as much information as they can, it should be the case that all that reach P are suffixes.

Figure 2 shows the process of establishing linear automaton, a demonstration of adding suffix state. The solid line refers to the pointer of son, while the dotted line refers to that of pre.

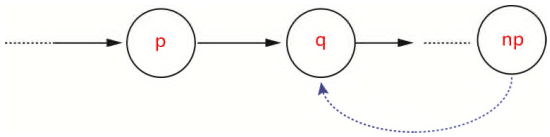

Fig.2. The process of the construction of the linear automata

step[q]>step[p]+1

Opposite to the above-mentioned situation, step[q]>step[p]+1 means that there could be other characters in between p and q, and that it cannot be ensured that when q is taken as the node that stores x, all the paths leading to q are suffixes of tx. The method applied on the former situation cannot be applied here. Based on the simplification method of the above automaton, the thesis establishes SA(Rev(X)) suffix automaton, and a linear structure as shown in Figure 3 can be established for the string “baabbaa”.

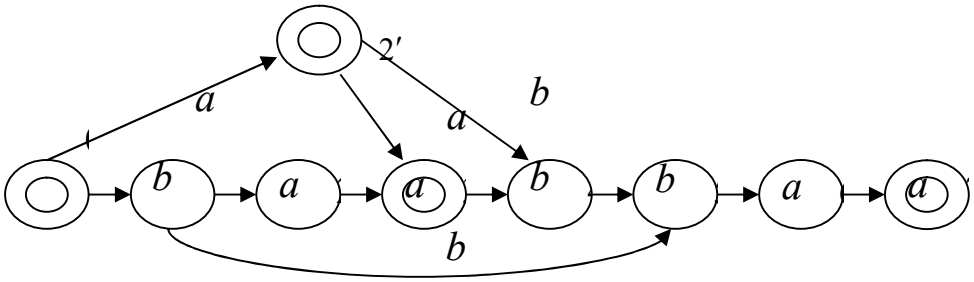

Fig.3. Simplified linear structure of least suffix automata

2.2 Finite State Automaton

Since the algorithm has been matched with one (or more) prefix of the pattern when the operation of the inverse suffix automaton stops, the forward finite state automaton starts its operation not from the traditional starting state, but the corresponding state of the biggest prefix that is already matched. The set of starting states excludes the traditional original state, because there is no need to use MFA for scanning if the prefix detected in the previous stage is ε. In practice, the detected length of prefix is usually seen to represent the state[7]. For instance, as Figure 4 shows, the character string aabbaab has linear automaton structure.

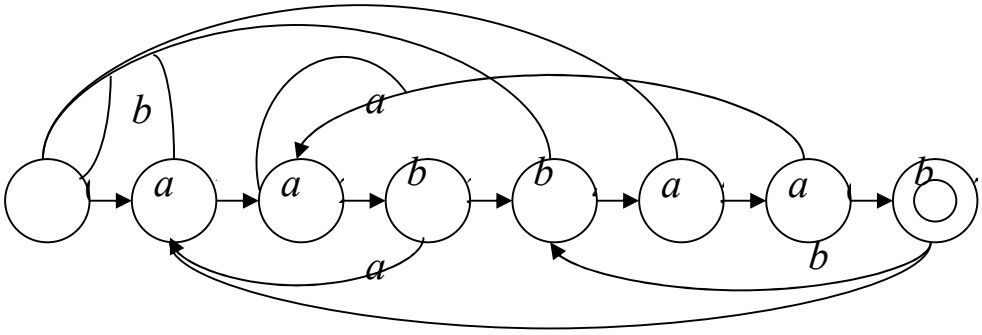

Fig.4 Simplified linear structure of finite state automata

2.3 LDM Algorithm

2.3.1 LDM Algorithm of String Matching

LDM Algorithm uses two types of automaton to do the window scanning: the finite state automaton with multi original states and the inverse least suffix automaton[8-10]. This algorithm mainly employs Dawg to scan the text and achieves the linearly worst time complexity. It divides the text string into [n/m] mutually overlapping windows, each with 2m-1 characters. Every window has m-1 characters identical with those in the previous and next windows, but the testing position appears only in the window it represents[7].

Take $y_{(k-1)m+1} \cdots y_{km} \cdots y_{(k+1)m-1}$ as an example:

(1) The inverse suffix automaton SA(Rev(X)) is used to scan the front window $y_{(k-1)m+1} \cdots y_{km}$ from back to front till it stops. When the automaton reaches the final state, the next position for scanning l, that is, from the length of suffix matched to the variable R, is recorded. When SA(Rev(X)) ends its operation, if R > 0, $y_{km-R+1} \cdots y_{km}$ will be the biggest suffix scanned by the algorithm in the previous window and the algorithm will continue; if R = 0, it means that the biggest suffix scanned is ε and as Figure 5 shows, the algorithm will jump backward and scan the next window.

(2) From the state R, MFA(X) is used to scan the back window $y_{km+1} \cdots y_{(k+1)m-1}$ from front to back. It needs at least m-R times of scanning before it reaches the final state, and there are still m-r+1 characters in the current window. If R < r, there will be no patterns appearing in the current window anymore, and the scanning ends. Then, when the automaton is in the final state, the current scanning position will be output. MFA scans from front to back, so if there are more than

one patterns in one window, the algorithm will always output the positions where patterns appear in the same order, which is shown in Figure 6.

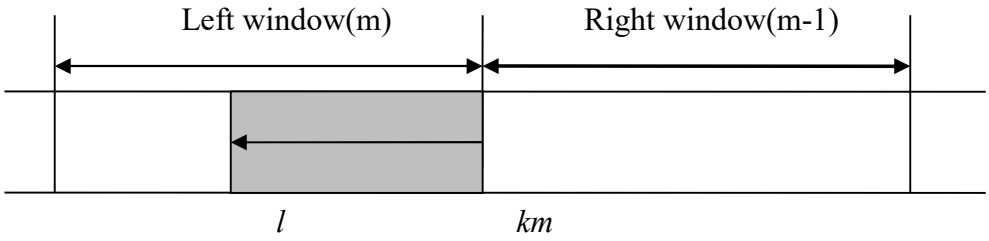

Fig.5 SA(Rev(X)) is used to search the pattern prefix from back to front till it stops

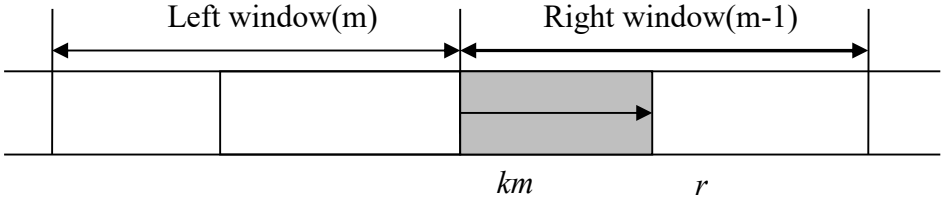

Fig.6 MFA(X) is used to search the pattern suffix from front to back till there is no other pattern string

2.3.2 LNDM Algorithm

LNDM algorithm makes some improvements based on LDM algorithm. It still employs only one machine word, but makes the mask moves inversely rather than makes the status words move. So, the status words will not be lost during the movements and all the matches in the current window will be saved. After the inverse scan, the machine word will be used again to start an automaton to do the forward scanning. This improvement enables LNDM algorithm to achieve the linearly worst time complexity and maintain the best average time complexity without asking for more resources[7].

The chart of masks is shown below:

$$B_i[c] = \begin{cases} 0 & if \quad X_i = c \\ 1 & otherwise \end{cases}$$

The main aim of establishing this chart of masks is to guarantee that when the mask moves left and carries the OR Operation with status words, the current status words will not be damaged. LNDM algorithm tries the windows $y_{(k-1)m+1} \ldots y_{km} \ldots y_{(k+1)m-1}$ successively, where k is from 1 to [n/m]. For one window, the scanning is divided into two stages.

The first stage: The status word L keeps still, while the mask B[l], after inverse translocation, carries OR Operation with the status word L. In this stage, the algorithm scans the window $y_{(k-1)m+1}\cdots y_{km}$ from back to front, as Figure 7 shows. Each time a character is read out, the following formula is employed to update the status word L:

$$L \leftarrow L \,|\, (B[y_{km-1}] << t)$$

Where t represents the iterative times during matching.

The scanning in this stage ends till $L = 1^{m-t}0^{t}$. After the scanning, if $L = 1^{m}$, it means no prefix has been matched and that the algorithm will move to the next window directly. Or, the algorithm will continue to the second stage.

When this stage ends, all the information about the matched prefixes will not be lost because of the operation of translocation, but be saved in the low order of the status word L, and L becomes the original state of the second stage.

The second stage: In this stage, the way of LNDM's operation is an obverse uncertain suffix automaton which is similar to a inverse uncertain suffix automaton that operates inversely. As Figure 8 shows, the algorithm scans the back window $y_{km+1}\cdots y_{(k+1)m-1}$ from front to back. Each time a character is read out, the following formula is employed to update the status word L:

$$L \leftarrow (L << 1) \,|\, B[y_{km+r}]$$

When $L_1 = 0$, a pattern that appears is reported. When $L = 1^{m}$, if there is no pattern appearing in the current window anymore, the algorithm will move on to the next window.

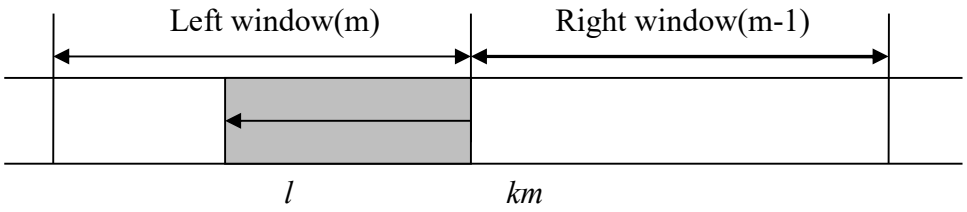

Fig.7 Inverse scanning till $L=1^{m}$, the status word L keeps still

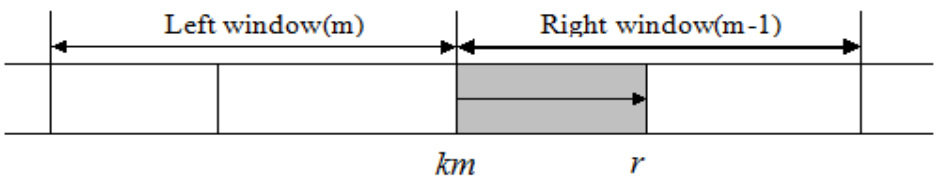

Fig.8 Obverse scanning till $L=1^{m}$, output current location when $L_1 = 0$

2.3.3 A Case of LNDM Index Search

Based on the algorithm model above, Matlab was applied to do a practical index search among the given text in the question on a platform where the dominant frequency is 3.4GHz,Intel CORE i7 to locate the positions of pattern string “ATACTA”.

It can be seen that it took only 0.0225 seconds to finish the search. The physical memory shown in the window is the sum of space occupied by MATLAB, which runs in the WINDOWS system (about 3700 MB), and the algorithm adopted in this paper. It shows that the memory space occupied by the application is about 150 MB, which means a decent ability of searching.

3. Discussion

3.1 Analysis of the Complexity

As Figure 9 shows, the time spent by the algorithm on establishing the index and searching for pattern strings is put into comparison with that of traditional algorithm.

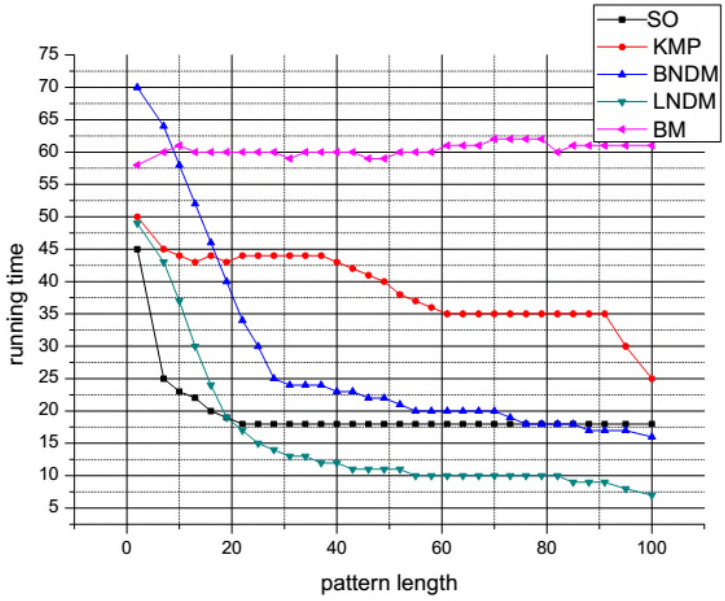

Fig.9 Comparisons of Time Spent on Searching Among Different Algorithms

It can be seen from the figure above that given a random text, x-axis represents the length of pattern while y-axis the average time spent on each 10M text by different algorithms. For short and patterns, LNDM is always the best algorithm.

3.2 Analysis of Memory

Storing text, the establishment of automaton and moving windows that scan the text are the main factors that influence memory. The relations between memory and the length of pattern string K are analyzed, and through the command of whos in Matlab environment, the specific memory used by the text is calculated and obtained. Hence, the changes of space memory corresponding to those of K value are shown. Figure 10 illustrates the functional image of the relations between K value and memory.

Space occupied by stored text: 156000000Byte $\div 1024 \div 1024 \times 2$=298M;

Space occupied by the establishment of automaton: K$\times 8Byte$;

Space occupied by the moving windows which scan the text: $8 \times (2k-1)$.

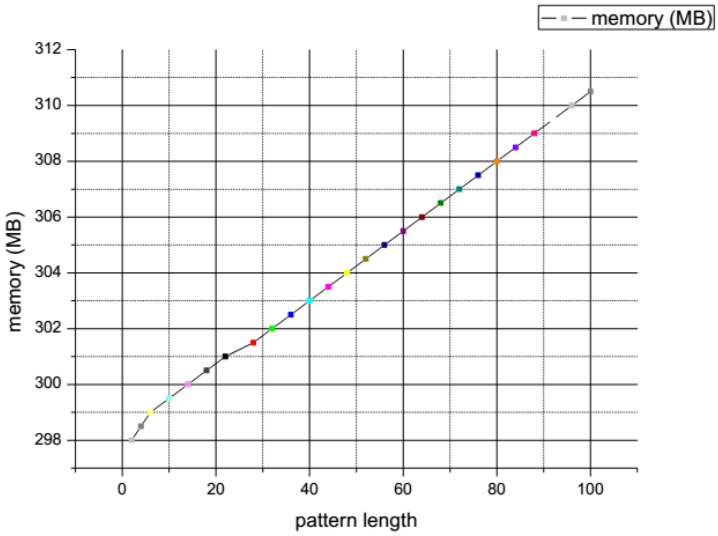

Fig.10 The Corresponding Relationship between the Length of DNA Strand and the Memory Space Occupied

The changing trend and related statistics shown in the figure help to prove that if the length of DNA strand is shorter than the machine word, the search model mentioned above is completely capable of locating DNA strands with different lengths.

3.3 Algorithm performance evaluation

According to the importance of from high to low arrangement: indexing query speed, memory, the range of K value supported, the time required for indexing, we evaluate the performance of index method. This paper design and implement a string matching algorithm test platform based on the four evaluation factors and fuzzy comprehensive evaluation model.

Assuming that the evaluation index set U=$\{u_1, u_2, \cdots, u_{11}\}$.U can be divided into four subsets: $U = \{U_1, U_2, U_3, U_4\}$, $U_i\ (i = 1,2,3,4)$: secondary comprehensive evaluation indicators, U_1 denote indexing query speed, U_2 denote memory, U_3 denote the range of K value supported, U_4 denote the time required for indexing.

Primary comprehensive evaluation indicators:

U_1 = {u_1(*Window average jump distance*), u_2(*Scanning speed*),
u_3(*Number of characters*), u_4(*Speed of update the bit mask*)}
U_2 = {u_5(*Upper limit of K value*), u_6(*Lower limit of K value*)}
U_3 = {u_7(*Text memory*), u_8(*Automatic memory*), u_9(*Search window memory*)}
U_4 = {u_{10}(*Building time used automata*), u_{11}(*Build time used window*)}

Assuming that comments sets :

V={v_1(worst), v_2(bad), v_3(middle), v_4(good), v_5(best)}.

The various factors of weight distribution as the fuzzy subset of U, $A = (a_1, a_2, a_3, a_4)$, a_i is the weight of the factor for i, $\sum_{i=1}^{n} a_i = 1 (a_i \geq 0)$. If the judge of factor i is fuzzy relations from U to V, $R_i(r_{i1}, r_{i2}, ...r_{im})$,the factors evaluation matrix R=$(r_{ij})_{n\times m}$.

Denote:

$$R_i = \begin{bmatrix} r_{11} & r_{22} & \cdots & r_{15} \\ r_{21} & r_{22} & \cdots & r_{25} \\ \cdots & \cdots & \cdots & \cdots \\ r_{n_i 1} & r_{n_i 2} & \cdots & r_{n_i 5} \end{bmatrix}$$

$\sum_{i=1}^{4} n_i = 11$, R_i as fuzzy relation from U_i to V, $r_{ij}(i = 1,2...; j = 1,2,3,4,5)$ as single factor u_i membership degree of evaluation v_i.

Making the distribution of the weight of each factor in U_i for $w_i = (w_{i1}, w_{i2}, w_{i3}, w_{i4})$, $\sum_{k=1}^{4} w_{ik} = 1$,and introducing comprehensive evaluation vector $B_i = w_i, R_i = \left(b_{i_1}, b_{i_2}, \cdots, b_{i_5}\right)$, i=1,2,3,4.

Building factor evaluation matrix about U :

$$R = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \end{bmatrix}$$

Distribution of the weight of each factor in U: $W=(W_1,W_2,W_3,W_4)$, get the secondary comprehensive evaluation vector:

$$B=W \qquad R=(b_1,b_2,\cdots,b_5),$$

$$b_j=\bigvee_{i=1}^{4}(W_i\wedge b_{ij})=\max_{1\le i\le 4}\{\min(W_i,b_{ij})\},\ j=1,2,\cdots,5,\ \ \sum_{k=1}^{4}W_k=1 .$$

b_j denote membership degree that search algorithm performance is rated v_k .According to the principle of maximum membership degree, If there is $k\in\{1,2,\cdots,5\}$, such that $B(v_k)=\max\{b_1,b_2,\cdots,b_5\}$,regard as that search algorithm performance evaluation in accordance with comments v_k .

Here we use the AHP method to determine the weight. Hierarchy is given according to evaluation index as follows:

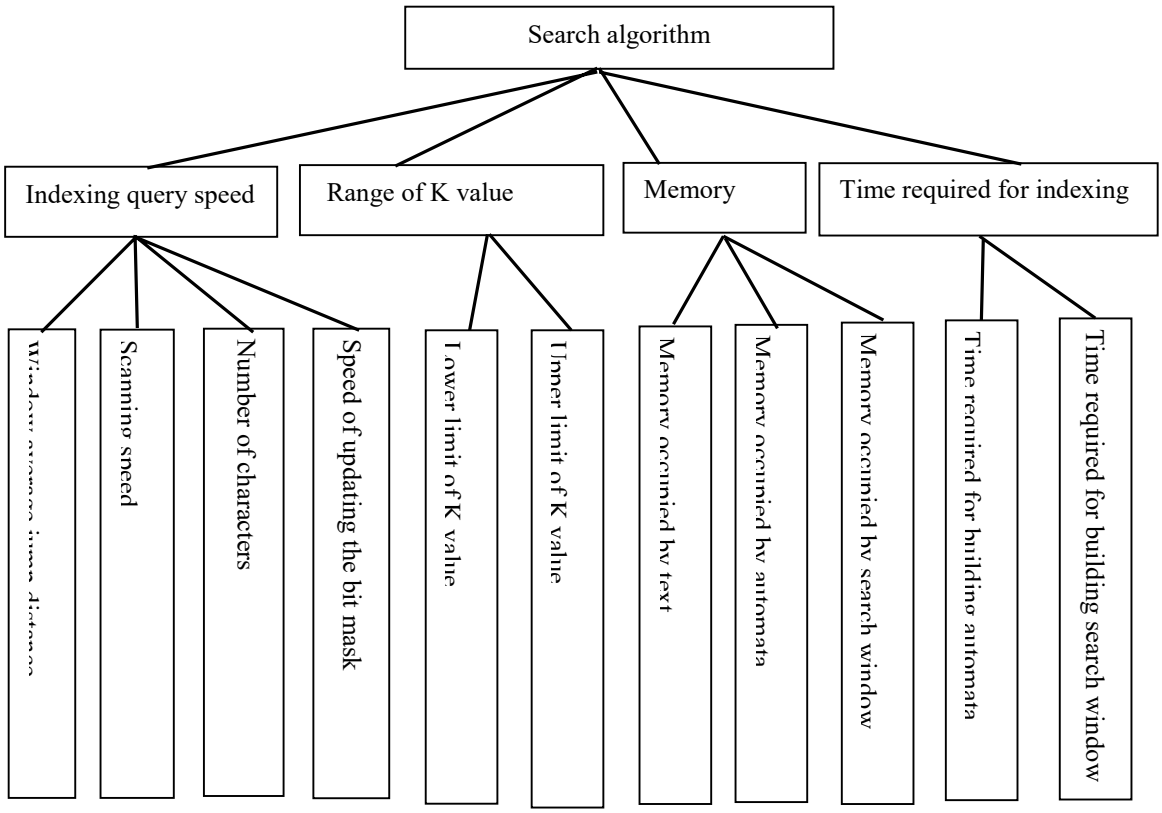

Fig. 11 Hierarchy

According to Aaaty scale to construct the judgment matrix of the first layer to the target layer: $A=\left(a_{ij}\right)_{4\times4}$, $a_{ij}>0, a_{ji}=\frac{1}{a_{ij}}, a_{ii}=1$. Based on the exponential method to calculate the maximum characteristic root $\lambda_{\max}$ and its corresponding eigenvector $W^0=\left(w_1^0, w_2^0, w_3^0, w_4^0\right)$, then do the consistency inspection, calculated

$$C.I.=\frac{\lambda_{\max}-4}{3} \qquad C.R.=\frac{C.R.}{R.I.}$$

here $R.I.=0.9$ .When $C.I.<0.1$,A satisfies the requirement of consistency, otherwise deal with A proper correction. If A satisfies the requirement of consistency, feature vector is normalized processing to get $W=\left(W_1, W_2, W_3, W_4\right)$, W is the weight distribution vector of each factor U_i . Continue to construct judgment matrix of the third to the second floor four indicators $D_j\left(j=1,2,3,4\right)$ .Using the above method can get the weight of each factor in the distribution vector $w_i=\left(w_{i_1}, w_{i_2}, \cdots, w_{i_{ni}}\right)\left(i=1,2,3,4\right)$.

For the algorithm established, we get a single factor evaluation matrix from U_i to V :

$$R_1=\begin{bmatrix}0.09 & 0.21 & 0.43 & 0.19 & 0.08\\ 0.32 & 0.41 & 0.11 & 0.09 & 0.07\\ 0.17 & 0.33 & 0.24 & 0.15 & 0.11\\ 0.00 & 0.12 & 0.19 & 0.22 & 0.47\end{bmatrix}$$

$$R_2=\begin{bmatrix}0.03 & 0.15 & 0.09 & 0.52 & 0.21\\ 0.18 & 0.37 & 0.32 & 0.13 & 0.00\end{bmatrix}$$

$$R_3=\begin{bmatrix}0.08 & 0.13 & 0.28 & 0.34 & 0.17\\ 0.00 & 0.07 & 0.28 & 0.43 & 0.22\\ 0.01 & 0.02 & 0.12 & 0.32 & 0.53\end{bmatrix}$$

$$R_4=\begin{bmatrix}0.07 & 0.05 & 0.26 & 0.49 & 0.13\\ 0.28 & 0.34 & 0.25 & 0.09 & 0.04\end{bmatrix}$$

The primary comprehensive evaluation vectors are:

$$B_1=w_1 \qquad R_1=\left(0.14, 0.21, 0.26, 0.22, 0.46\right)$$

$$B_2=w_2 \qquad R_2=\left(0.18, 0.25, 0.25, 0.52, 0.21\right)$$

$$B_3=w_3 \qquad R_3=\left(0.08, 0.13, 0.28, 0.43, 0.22\right)$$

$$B_4 = w_4 \qquad R_4 = (0.28, 0.34, 0.26, 0.34, 0.13)$$

B_i can be used to form the single factor evaluation matrix R about U. The secondary comprehensive evaluation vectors are:

$$B = w \qquad R = (0.14, 0.21, 0.27, 0.43, 0.27).$$

According to the principle of maximum membership, the search method used with good performance, comply with the important attributes required of the index.

4. Conclusion

This paper utilizes automaton's superior performance in identifying character string, establishes two classical suffix automata, that is, the smallest suffix automaton and the finite state automaton. The automata are simplified into linear structures in order to economize memory space and enable them to match pattern strings in two directions from the middle point of the window. Finally, after moving the windows to cover the whole text and updating the information in bitmasks, all positions where the pattern strings that are searched for appear in the text are output. This paper also employs the automaton system to establish the index model of LNDM algorithm, which can quickly locate the positions of target short DNA strands in the DNA bank.

Acknowledgement

Supported by Youth Foundation of Changchun University of Science and Technology (Grant No. XQNJJ-2013-01).

References

1. Ela Hunt, Malcolm P. Atkinson, Robert W. Irving, Database indexing for large DNA and protein sequence collections,2002,The VLDB Journal , vol.11, no. 3, pp. 256-271.
2. Jung-Im Won, Jee-Hee Yoon, Sanghyun Park, Sang-Wook Kim, A Novel Indexing Method for Efficient Sequence Matching in Large DNA Database Environment, 2005, Advances in Knowledge Discovery and Data Mining Lecture Notes in Computer Science ,vol.3518, pp. 203-215.
3. Yingxin Hu, Zhaohui Qi, Lijuan Zheng, Wenfeng Zhou. Similarity Analysis of DNA Sequences Based on K-word, 2014, Proceedings of 2014 IEEE International Conference on Progress in Informatics and Computing, 2014, Shanghai, pp.621-625.

4. Chuan-Wen Li, Yu Gu, Ge Yu, Bonghee Hong. Aggressive Complex Event Processing with Confidence over Out-of-Order Streams , 2011, Journal of computer science and technology , vol. 26, no.4,pp. 685-696 .
5. Changil Choe, Song Han, Dang Van Hung.Approximate Model Checking of Real-time Systems for Linear Duration Invariants ,2012, Proceedings of 2012 International Conference on Applied Informatics and Communication, 2012, Shenzhen, pp.16-21.
6. Jianhua Zhao,Van Hung Dang,Checking timed automata for linear duration properties, 2000, Journal of Computer Science and Technology ,vol.15,no.5,pp.424-428.
7. Zhou Jiangtao, Research and implementation of exact string matching algorithms, 2008, Harbin Institute of Technology.
8. Sun Liangxu, Li Linlin.Improve Wu-Manber Algorithm for Multiple Pattern Matching Time Efficiency Optimization, 2012, International Journal on Advances in Information Sciences and Service Sciences, vol. 4, no.20, pp.220-228.
9. Ko Kusudo, Fumihiko Ino, Kenichi Hagihara.A bit-parallel algorithm for searching multiple patterns with various lengths, 2015, Journal of Parallel and Distributed Computing, no.76, pp. 49-57.
10. Rajesh Prasad, Suneeta Agarwal, Ishadutta Yadav, Bharat Singh, Efficient bit-parallel multi-patterns string matching algorithms for limited expression, 2010, Proceedings of the 3rd Bangalore Annual Compute Conference, Compute , Bangalore, India, 2010, pp.22-23.