A FAST LONGEST COMMON SUBSEQUENCE ALGORITHM FOR BIOSEQUENCES ALIGNMENT

Wei Liu [1,*], Lin Chen [2,3]

[1] *Institute of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210093, China*

[2] *Department of Computer Science, Yangzhou University, Yangzhou 225009, China*

[3] *State Key Lab of Novel Software Technology, Nanjing University, Nanjing 210093, China*

[*] *Corresponding author, Address: P.O. Box 274, Institute of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, 29 Yudao ST., Nanjing, 210093, P. R. China, Email: yzliuwei@126.com*

Abstract: Searching for the longest common substring (LCS) of biosequences is one of the most important tasks in Bioinformatics. A fast algorithm for LCS problem named FAST_LCS is presented. The algorithm first seeks the successors of the initial identical character pairs according to a successor table to obtain all the identical pairs and their levels. Then by tracing back from the identical character pair at the largest level, the result of LCS can be obtained. For two sequences X and Y with lengths n and m, the memory required for FAST_LCS is $\max\{8*(n+1)+8*(m+1),L\}$, here L is the number of identical character pairs and time complexity of parallel implementation is $O(|\mathrm{LCS}(X,Y)|)$, here, $|\mathrm{LCS}(X,Y)|$ is the length of the LCS of X,Y. Experimental result on the gene sequences of *tigr* database using MPP parallel computer Shenteng 1800 shows that our algorithm can get exact correct result and is faster and more efficient than other LCS algorithms.

Keywords: bioinformatics; longest common subsequence; identical character pair

Liu, W. and Chen, L., 2008, in IFIP International Federation for Information Processing, Volume 258; Computer and Computing Technologies in Agriculture, Vol. 1; Daoliang Li; (Boston: Springer), pp. 61–69.

1. INTRODUCTION

Biological sequence (Bailin Hao et al., 2000) can be represented as a sequence of symbols. When biologists find a new sequence, they want to know what other sequences it is most similar to. Sequence comparison (Edmiston E.W. et al., 1988)has been used successfully to establish the link between cancer-causing genes and a gene evolved in normal growth and development. One way of detecting the similarity of two or more sequences is to find their longest common subsequence (LCS).

The longest common subsequence problem is to find a substring that is common to two or more given strings and is the longest one of such strings. Presented in 1981, Smith-Waterman algorithm (Smith T.F. et al., 1990) was a well known LCS algorithm which was evolved by the Needleman-Wunsch (Needleman, S.B. et al., 1970) algorithm, and can guarantee the correct result. Aho et al. (A. Aho et al., 1976) gave a lower bound of $O(mn)$ on time for the LCS problem using a decision tree model. It is shown in (O. Gotoh, 1982) that the problem can be solved in $O(mn)$ time using $O(mn)$ space by dynamic programming. Mayers and Miller (E.W. Mayers et al., 1998) use the skill proposed by Hirschberg (D.S. Hirschberg, 1975) to reduce the space complexity to $O(m+n)$ on the premise of the same time complexity. To further reduce the computation time, some parallel algorithms (Y. Pan et al., 1998, Jean Frédéric Myoupo et al., 1999, L. Bergroth et al., 2000, A. Aggarwal et al., 1988) have been proposed for the LCS problem on different computational models. On CREW-PRAM model, Aggarwal (A. Aggarwal et al.,1988) and Apostolico et al. (A. Apostolico et al., 1990) independently proposed an $O(\log m \log n)$ time algorithm using $mn/\log m$ processors. Many parallel LCS algorithms have also been proposed on systolic arrays. Robert et al. (K. Nandan Babu et al., 1997) proposed a parallel algorithm with $n+5m$ steps using $m(m+1)$ processing elements. Freschi and Bogliolo (V. Freschi et al., 2000) addressed the problem of computing the LCS between run-length-encoded (RLE) strings. Their algorithm requirs $O(m+n)$ steps on a systolic array of $M+N$ processing elements, where M and N are the lengths of the original strings and m and n are the number of runs in their RLE representation.

In this paper, we present a fast algorithm named FAST_LCS for LCS problem. The algorithm first seeks the successors of the initial identical character pairs according to a successor table to obtain all the identical pairs and their levels. Then by tracing back from the identical character pair at the largest level, the result of LCS can be obtained. For two sequences X and Y with lengths n and m, the memory required for FAST_LCS is $\max\{8*(n+1)+8*(m+1), L\}$, here L is the number of identical character pairs and time complexity of parallel implementation is $O(|LCS(X,Y)|)$, here, $|LCS(X,Y)|$ is the length of the LCS of X,Y. Experimental result on the gene

sequences of *tigr* database using MPP parallel computer Shenteng 1800 shows that our algorithm can get exact correct result and is faster and more efficient than other LCS algorithms.

2. THE IDENTICAL CHARACTER PAIR AND ITS SUCCESSOR TABLE

Let X ($x_1, x_2, \ldots, x_n$), $Y = (y_1, y_2, \ldots, y_m)$ be two biosequences, where x_i , $y_i \in \{A,C,G,T\}$. We can define an array CH of the four characters so that $CH(1)$="A", $CH(2)$="C", $CH(3)$="G " and $CH(4)$="T ". To find their longest common subsequence, we first build the successor tables of the identical characters for the two strings. The successor tables of X and Y are denoted as TX and TY which are $4*(n+1)$ and $4*(m+1)$ two dimensional arrays. $TX(i, j)$ is defined as follows.

Definition 1. For the sequence $X = (x_1, x_2, \ldots, x_n)$, its successor table TX of identical character is defined as:

$$TX(i,j) = \begin{cases} \min\{k \mid k \in SX(i,j)\} & SX(i,j) \neq \phi \\ - & \text{otherwise} \end{cases} \quad (1)$$

Here, $SX(i, j)=\{k| x_k=CH(i), k>j\}$, $i = 1,2,3,4$, $j = 0,1,\ldots n$. It can be seen from the definition that if $TX(i, j)$ is not "-", it indicates the position of the next character identical to $CH(i)$ after the jth position in sequence X. If $TX(i, j)$ is equal to "-", it means there is no character $CH(i)$ after the jth position.

Example 1. Let X ="$T\ G\ C\ A\ T\ A$", Y ="$A\ T\ C\ T\ G\ A\ T$ ". Their successor tables TX and TY are:

TX:

i	$CH(i)$	0	1	2	3	4	5	6
1	A	4	4	4	4	6	6	-
2	C	3	3	3	-	-	-	-
3	G	2	2	-	-	-	-	-
4	T	1	5	5	5	5	-	-

TY:

i	$CH(i)$	0	1	2	3	4	5	6
1	A	4	4	4	4	6	6	-
2	C	3	3	3	-	-	-	-
3	G	2	2	-	-	-	-	-
4	T	1	5	5	5	5	-	-

Definition 2. For the sequences X and Y, if $x_i = y_j$, we call them an identical character pair of X and Y, and denote it as (i, j). The set of all the identical character pairs of X and Y is denoted as $S(X, Y)$.

Definition 3. Let (i, j) and (k, l) be two identical character pairs of X and Y. If $i<k$ and $j<l$, we call (i, j) a predecessor of (k, l), or (k, l) a successor of (i, j), and denote them as $(i, j)<(k, l)$.

Definition 4. Let $P(i, j) = \{(r, s)|(i, j) < (r, s), (r, s) \in S(X,Y)\}$ be the set of all the successors of identical pair (i, j), if $(k, l) \in P(i, j)$ and there is no

$(k', l') \square P(i, j)$ satisfying the condition: $(k', l') < (k, l)$, we call (k, l) the direct successor of (i, j), and denoted it as $(i, j) \prec (k, l)$.

Definition 5. If an identical pair $(i, j) \square S(X,Y)$ and there is no $(k, l) \square S(X,Y)$ so that $(k, l) < (i, j)$, we call (i, j) an initial identical pair.

Definition 6. For an identical pair $(i, j) \square S(X,Y)$, its level is defined as follows:

$$level(i,j) = \begin{cases} 1 & \text{if } (i,j) \text{ is an initial identical character pair} \\ \max\{level(k,l)+1 \mid (k,l) < (i,j)\} & \text{otherwise} \end{cases} \quad (2)$$

From the definitions above, the following lemma can be easily deduced:

Lemma 1. Denote the length of the longest common subsequence of X, Y as |LCS(X, Y)|, then |LCS(X, Y)|=max{level (i, j) |$(i, j) \in S(X, Y)$}.

Proof of Lemma 1 is omitted due to space limitation.

3. THE OPERATIONS OF PRODUCING SUCCESSORS AND PRUNING

For an identical character pair $(i, j) \square S(X, Y)$, the operation of producing all its direct successors is as follows:

$$(i,j) \to \{(TX(k,i), TY(k,j)) \mid k = 1,2,3,4, TX(k,i) \neq '-' \text{ and } TY(k,j) \neq '-'\} \quad (3)$$

From (3) we can see that this operation is to couple the elements of the ith column of TX and the jth column of TY to get the pairs.

Lemma 2. For an identical character pair (i, j), the method illustrated above can produce all its successors.

Proof of Lemma 2 is omitted due to space limitation.

In such process of generating the successors, pruning technique can be implemented to remove the identical pairs so as to reduce the searching space and accelerate the speed of process. These prune operations are based on the following theorems.

Theorem 1. If two identical character pairs (i, j) and (k, l) generated at the same time step satisfy $(k, l)>(i, j)$, then (k, l) can be pruned without affecting the algorithm to get the longest common subsequence of X and Y.

Theorem 2. If on the same level, there are two identical character pairs (i_1, j) and (i_2, j) satisfying $i_1 < i_2$, then (i_2, j) can be pruned without affecting the algorithm to get the longest common subsequence of X and Y.

Proof of Theorem 1 and 2 is omitted due to space limitation.

4. FRAMEWORK OF THE ALGORITHM AND COMPLEXITY ANALYSIS

Based on the operations mentioned above, we present a fast parallel longest common subsequence algorithm FAST_LCS. The algorithm first begins with the initial identical character pairs, then continuously searches for their successors using the successor tables. In this phase, the pruning technology is implemented to discard those search branches that obviously can't obtain the optimum solution so as to reduce the search space and speed up the process of searching. In the algorithm, a table called *pairs* is used to store the identical character pairs obtained in the algorithm. In the table *pairs*, each record takes the form of (*k*, *i*, *j*, *level*, *pre*, *state*) where the data items denote the index of the record, the identical character pair (*i*, *j*), its level, index of its direct predecessor and its current state. Each record in *pairs* has two states. For the identical pairs whose successors have not been searched, it is in *active* state, otherwise it is in *inactive* state. In every step of search process, the algorithm searches for the successors of all the identical pairs in *active* state in parallel. Repeat this search process until there is no identical pair in *active* state in the table. The phase of tracing back starts from the identical pairs with the maximal level in the table, and traces back according to the *pred* of each identical pair. This tracing back process ends when it reaches an initial identical pair, and the trail indicates the longest common subsequence. If there are more than one identical pair with the maximal level in the table, the tracing back procedure for those identical pairs can be carried out in parallel and several longest common subsequences can be obtained concurrently. The framework of the algorithm FAST_LCS is as follows:

Algorithm-FAST_LCS (*X*,*Y*)
Input *X* and *Y*: Sequences with lengths of *m* and *n* respectively;
Output LCS: The longest common subsequence of *X*,*Y*;
Begin

1. Build tables *TX* and *TY*;
2. Find all the initial identical character pairs: (*TX*(*k*, 0), *TY*(*k*, 0)), *k*=1,2,3,4;
3. Add the records of the initial identical pairs (*k*, *TX*(*k*, 0), *TY*(*k*, 0), 0, ϕ, *active*), *k*=1,2,3,4 to the table *pairs*.
   /* For all the initial identical pairs, their *level*=1, *pre*=ϕ and *state*=*active**/
4. Repeat

For all *active* identical pairs (*k*, *i*, *j*, *level*, *pre*, *active*) in *pairs* parallel-do

Produce all the successors of (*k*, *i*, *j*, *level*, *pre*, *active*).

For each identical character pair (*g*, *h*)in the successor set of (*k*, *i*, *j*, *level*, *pre*, *active*), a new record (*k'*, *g*, *h*, *level*+1, *k*, *active*) is generated and inserted into the table *pairs*.

Change the state of (*k*, *i*, *j*, *level*, *pre*, *active*) into *inactive*.

End for

Use prune operation on all the successors produced in this level to remove all the redundant identical pairs from table *pairs*.

5. Until there is no record in *active* state in table *pairs*.
6. Compute r = the maximal level in the table pairs.
7. For all identical pairs (*k*, *i*, *j*, *r*, *l*, *inactive*) with *level* = *r* in *pairs* parallel-do

Pred = *l*; $LCS(r) = x_i$.

While *pred* $\neq \phi$ do

7.1.1 get the *Pred*-th record (*prel*, *g*, *h*, *r'*, *l'*, *inactive*) from table *pairs*.

7.1.2 *Pred* = *l'*; $LCS(r') = x_g$.

7.3 end while

End.

Assume that the number of the identical character pairs of *X*, *Y* is *L*. In our algorithm, the time complexity for sequentially executing of the algorithm FAST_LCS (*X*, *Y*) is $O(L)$ and the storage complexity of our algorithm is $\max\{8*(n+1)+8*(m+1), L\}$. In parallel implementation of the algorithm, the time complexity of parallel computing is $O(|\mathrm{LCS}(X,Y)|)$, here, $|\mathrm{LCS}(X,Y)|$ is the length of the longest common subsequence of *X*, *Y*.

5. EXPERIMENTAL RESULTS

5.1 The results of sequential computing-two sequences

We test our algorithm FAST_LCS on the rice gene sequences of *tigr* database and compare the performance of FAST_LCS with that of Smith-Waterman algorithm and FASTA algorithm which are currently the most widely used LCS algorithms.

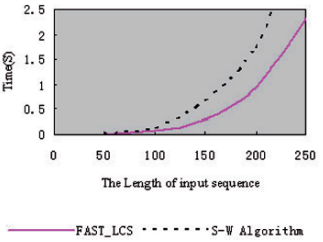

Fig. 1. Comparison of the computation time of FAST_LCS with that of Smith-Waterman algorithm

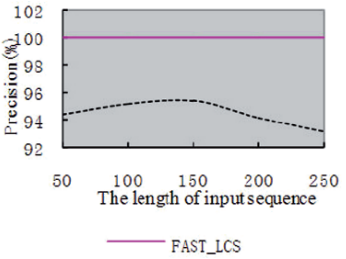

Fig. 2. Comparison of the precision of FAST_LCS with that of FASTA using the same computation time

Fig. 1 shows the comparison of the computation time of our algorithm and that of Smith-Waterman algorithm. From the figure, we see that our algorithm is obviously faster than Smith-Waterman algorithm for sequences sets of all different lengths, especially when the length of sequences become greater than 150. This means our algorithm is much faster and more efficient than Smith-Waterman's for LCS problem of long sequences.

We also compare the precision of our algorithm with that of FASTA on the premise of the same computing time. Here precision is defined as:

$$\text{Precision} = \frac{\text{The length of the common subsequence computed by the algorithm}}{\text{The length of the longest common subsequence in correct match}}$$

From Fig. 2, we can see that our algorithm can obtain exactly correct result no matter how long the sequence could be, while the precision of FASTA declines when the length of the sequences is increased. Therefore the precision of our algorithm is higher than FASTA algorithm.

5.2 The results of parallel computing

We also test our algorithm on the rice gene sequence from *tigr* database on the massive parallel processors Shenteng 1800 using MPI (C bounding). In the parallel implementation of our algorithm FAST_LCS, the identical character pairs in *active* state are assigned and processed in different processors. The experimental results by using different numbers of processors are shown in Fig. 3. From Fig. 3, we can see that the computation speed will become faster as the number of processors increases. But the speedup will slow down when the number of processors is larger than 6. Because of the overhead of communication between processors which increases the total time of the algorithm, the speedup of our algorithm can not increase linearly with the increasing of processors exactly. This is in conformity with the Amdahl's Law.

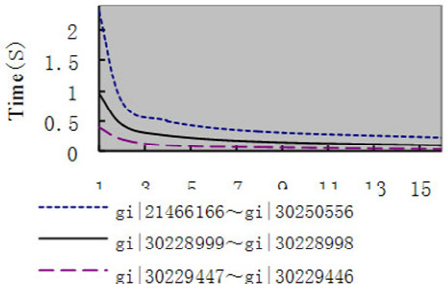

Fig. 3. Parallel computational time of FAST_LCS using different processor numbers

6. CONCLUSION

On the premise of guaranteeing precision, this paper present a parallel longest common subsequence algorithm FAST_LCS based on the identical character pair to improve the speed of LCS problem. Our algorithm first seeks the successors of the initial identical character pairs according to a successor table to obtain all the identical pairs and their levels. Then by tracing back from the identical character pair with the largest level, the result of LCS can be obtained. For two sequences X and Y with length n and m, the memory required for FAST_LCS is $\max\{4*(n+1)+4*(m+1),L\}$, here L is the number of identical character pair and time complexity of parallel computing is $O(|\mathrm{LCS}(X,Y)|)$, here $|\mathrm{LCS}(X,Y)|$ is the length of the LCS of X,Y. Experimental result on the gene sequences of *tigr* database using MPP parallel computer Shenteng 1800 shows that our algorithm can get exact correct result and is faster and more efficient than other LCS algorithms.

ACKNOWLEDGEMENTS

This research was supported in part by Chinese National Natural Science Foundation under grant No. 60673060, and Natural Science Foundation of Jiangsu Province under contract BK2005047.

REFERENCES

A. Aggarwal and J. Park, 1988, Notes on Searching in Multidimensional Monotone Arrays, Proc. 29th Ann. IEEE Symp. Foundations of Comput. Sci. pp. 497-512.

A. Aho, D. Hirschberg, and J. Ullman, 1976, Bounds on the Complexity of the Longest Common Subsequence Problem, J. Assoc. Comput. Mach., Vol. 23, No. 1, 1976, pp. 1-12.

A. Apostolico, M. Atallah, L. Larmore, and S. Mcfaddin, 1990, Efficient Parallel Algorithms for String Editing and Related Problems, SIAM J. Computing, Vol. 19, pp. 968-988.

Bailin Hao, Shuyu Zhang, 2000, The manual of Bioinformatics, Shanghai science and technology publishing company.

D.S. Hirschberg, 1975, A Linear Space Algorithm for Computing Maximal Common Subsequences, Commun. ACM, Vol. 18, No. 6, pp. 341-343.

E.W. Mayers, W. Miller, 1998, Optimal Alignment in Linear Space, Comput. Appl. Biosci. Vol. 4, No. 1, pp. 11-17.

Edmiston E.W., Core N.G., Saltz J.H, et al., 1988, Parallel processing of biological sequence comparison algorithms. International Journal of Parallel Programming, Vol. 17, No. 3, pp. 259-275.

Jean Frédéric Myoupo, David Seme, 1999, Time-Efficient Parallel Algorithms for the Longest Common Subsequence and Related Problems, Journal of Parallel and Distributed Computing, Vol. 57, No. 2, pp. 212-223.

K. Nandan Babu, Wipro Systems, and Sanjeev Saxena, 1997, Parallel Algorithms for the Longest Common Subsequence Problem, 4th International Conference on High Performance Computing, December 18-21, 1997 - Bangalore, India.

L. Bergroth, H. Hakonen, and T. Raita, 2000, A survey of longest common subsequence algorithms, Seventh International Symposium on String Processing Information Retrieval, pp. 39-48.

Needleman, S.B. and Wunsch, C.D., 1970, A general method applicable to the search for similarities in the amino acid sequence of two proteins, J. Mol. Biol., Vol. 48, No. 3, pp. 443-453.

O. Gotoh, 1982, An improved algorithm for matching biological sequences, J. Molec. Biol. Vol. 162, pp. 705-708.

Smith T.F., Waterman M.S. 1990, Identification of common molecular subsequence. Journal of Molecular Biology, Vol. 215, pp. 403-410.

V. Freschi and A. Bogliolo, 2004, Longest common subsequence between run-length-encoded strings: a new algorithm with improved parallelism, Information Processing Letters, Vol. 90, No. 4, pp. 167-173.

Y. Pan, K. Li, 1998, Linear Array with a Reconfigurable Pipelined Bus System - Concepts and Applications, Journal of Information Science, Vol. 106, pp. 237-258.