Available online at www.sciencedirect.com
ScienceDirect
Information Processing Letters 102 (2007) 99–103
Information Processing Letters
www.elsevier.com/locate/ipl

Dynamic programming algorithms for the mosaic longest common subsequence problem ☆

Kuo-Si Huang, Chang-Biau Yang *, Kuo-Tsung Tseng, Yung-Hsing Peng, Hsing-Yen Ann

Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan

Received 11 September 2006; received in revised form 7 November 2006; accepted 10 November 2006

Available online 11 December 2006

Communicated by L. Boasson

Abstract

The longest common subsequence (LCS) problem can be used to measure the relationship between sequences. In general, the inputs of the LCS problem are two sequences. For finding the relationship between one sequence and a set of sequences, we cannot apply the traditional LCS algorithms immediately. In this paper, we define the mosaic LCS (MLCS) problem of finding a mosaic sequence C, composed of repeatable k sequences in source sequence set S, such that the LCS of C and the target sequence T is maximal. Based on the concept of break points in sequence T, we first propose a divide-and-conquer algorithm with $O(n^2m|S| + n^3 \log k)$ time for solving this problem, where n and m are the length of T and the maximal length of sequences in S, respectively. Furthermore, an improved algorithm with $O(n(m + k)|S|)$ time is proposed by applying an efficient preprocessing for the MLCS problem.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Dynamic programming; Bioinformatics; Longest common subsequence; Mosaic sequence; Design of algorithms

1. Introduction

The longest common subsequence (LCS) problem is a classical one in computer science and it has been widely discussed over three decades [3]. The LCS can be applied to many areas, such as file comparison, speech recognition, and especially bioinformatics [2,9]. In bioinformatics, the LCS of a set of sequences can be regarded as the identity of the input sequences and we can reconstruct an alignment based on the LCS. In addition, the LCS can help us find out the segments with biological meanings such as motifs, promoters, reception sites, and conserved regions. In general, the LCS is used to find the relationship of two sequences. If we desire to know the relationship between one sequence and a set of sequences, the original LCS algorithms cannot return the relationship directly and hence a new algorithm should be designed. For example, given a set of short and meaningful sequences, one wants to find out the minimum of these short sequences to cover a long query sequence above a predefined coverage threshold. Hereby, one can get useful information about the query sequence of applications such as recognizing the rela-

☆ This research work was partially supported by the National Science Council of Taiwan under NSC-95-2221-E-110-084.

* Corresponding author.
E-mail address: cbyang@cse.nsysu.edu.tw (C.-B. Yang).

0020-0190/$ – see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.ipl.2006.11.006

tionship between short sequences, locating key features, and compressing this long sequence.

In this paper, we discuss the *mosaic LCS* (MLCS) problem. Given a target sequence T and a set S of source sequences, the MLCS problem is to find a *mosaic sequence* C, composed of repeatable k sequences in S, such that the LCS of C and T is maximal. It is indiscreet by using the brute-force strategy to solve this problem. Based on the concept of break points in sequence T, we propose an algorithm with $O(n^2m|S| + n^3 \log k)$ time for solving the MLCS problem, where n, m, and $|S|$ are the length of T, the maximal length of sequences in S, and the cardinality of S, respectively. Furthermore, by applying the S-table information [7,6], we propose an improved algorithm with $O(n(m+k)|S|)$ time for solving the problem.

2. Definitions

For a sequence T, over a finite alphabet set Σ, a subsequence of T is a sequence obtained from T by deleting zero or more symbols. The *LCS problem* is defined as finding the common subsequence T' of given sequences such that T' is the longest one among all possible common subsequences. If the number of input sequences is not fixed, the problem is NP-complete even over binary alphabets [8]. In general, many researches and applications focus on the LCS problem for only two input sequences [2,9] and there exist $O(n^2)$ algorithms in the worst case for the LCS problem of two sequences. For two sequences $S_1 = \texttt{agactagtc}$ and $S_2 = \texttt{tagtcacg}$, sequences `agc`, `agtc`, `agacg`, `agtag`, and `agtac` are some common subsequences of S_1 and S_2; in particular, `agacg`, `agtag`, and `agtac` are all the LCS's of S_1 and S_2, denoted as $LCS(S_1, S_2)$, with maximal length 5. Note that in this paper, $LCS(S_1, S_2)$ may be used to represent either the length value of $LCS(S_1, S_2)$ or the string of $LCS(S_1, S_2)$ alternatively if there is no ambiguity of its meaning.

Some more notations are given as follows. $|S|$ denotes the cardinality or the length of S if S is a set or a sequence. Substring $T[p, q]$ represents the contiguous segment of T from positions p to q. In addition, $T[p, q] = \emptyset$, the empty string, if $p > q$. We use $C = C_1C_2 \dots C_k$ to represent the mosaic sequence or the composite sequence composed of C_i's, $1 \leqslant i \leqslant k$. The notation $C\langle i, j\rangle$ means the partial contiguous sequences of C, i.e., $C\langle i, j\rangle = C_iC_{i+1}C_{i+2} \dots C_{j-1}C_j$. Also, $C\langle i, j\rangle$ means an empty sequence if $i > j$. A bold symbol or number represents a vector. We suppose the lengths of T and the longest sequence in S are n and m, respectively.

The *mosaic longest common subsequence* (MLCS) problem is defined as follows. Given a target (query) sequence T, a mosaic number k, $k \geqslant 1$, and a set of source sequences S, find a k-mosaic (k-concatenate) sequence $C = C_1C_2 \dots C_k$, $C_i \in S$ and $1 \leqslant i \leqslant k$, such that $LCS(T, C)$ is maximal. For example, suppose $T = \texttt{agactagtc}$, $k = 3$, and $S = \{S_1 = \texttt{agc}, S_2 = \texttt{act}, S_3 = \texttt{aatg}, S_4 = \texttt{ttcg}\}$. We can find $C = \texttt{agc act agc}$ such that $LCS(T, C) = \texttt{agactagc}$ with length 8. If $k = 4$, we can construct another $C = \texttt{agc act agc ttcg}$ such that $LCS(T, C) = \texttt{agactagtc}$ with length 9. We note that if $k = 1$, the MLCS problem is to find $\max\{LCS(T, S_i) \mid S_i \in S\}$ and it is equivalent to the traditional sequence searching problem. In addition, the MLCS problem has better applications and meanings in the case of $m < n$.

3. Algorithms

The solution of the MLCS problem can be obtained clearly by computing $LCS(T, C) = \max\{LCS(T, P_i)\}$ for all $|S|^k$ possible mosaic sequences $P_i \in S^k$, where $|P_i| = O(mk)$. The time complexity of this brute-force algorithm is $O(mnk|S|^k)$ and it is exponential to the mosaic number k. If the mosaic number k and the cardinality of S are large, this algorithm could not obtain a solution in feasible time. For solving the MLCS problem more efficiently, we use the *break points* in T to improve our algorithm. The essential concept of break point is included in the divide-and-conquer approach proposed by Hirschberg for finding the LCS [3].

Lemma 1. [3] *Given sequences T and $C = C_1C_2$, we can find a position r in T such that $LCS(T, C) = LCS(T[1, r], C_1) + LCS(T[r + 1, n], C_2)$.*

Komatsoulis and Waterman solved a specific version of the MLCS problem with the restriction of the mosaic number $k = 2$ in $O(mn|S|)$ time [4] by applying Lemma 1 implicitly. In addition, their algorithm is a chimeric sequence alignment algorithm with $k = 2$, and it supposes that there are two disjoint sets S_A and S_B, where $C_1 \in S_A$ and $C_2 \in S_B$ [4]. Komatsoulis and Waterman applied it to detect chimeric 16S rRNA artifacts in biology [5]. We can regard the 2-chimeric LCS problem as to find the best chimeric sequence $C = C_1C_2$ such that $LCS(T, C)$ is maximal where $C_1 \in S$ and $C_2 \in S \setminus \{C_1\}$. Based on Lemma 1, we conclude the following results.

Lemma 2. *Given a sequence T and a composite sequence C with the mosaic number k ($k \geqslant 2$), for*

any k', $1 \leqslant k' \leqslant k$, there exists a break point r such that $LCS(T[1,n], C\langle 1,k\rangle) = LCS(T[1,r], C\langle 1,k'\rangle) + LCS(T[r+1,n], C\langle k'+1,k\rangle)$.

Proof. If a composite sequence C is given, C can be separated into prefix $C\langle 1,k'\rangle$ and suffix $C\langle k'+1,k\rangle$ for any k', $1 \leqslant k' \leqslant k$. It is clear that this lemma holds based on Lemma 1, the concept of finding the break point in a divide-and-conquer approach [3]. □

Lemma 3. *Given a sequence T, a set S of sequences and the mosaic number k ($k \geqslant 2$), we can obtain the mosaic sequence $C = C\langle 1,k\rangle$ by finding a vector of break points $\mathbf{r} = (r_0, r_1, \ldots, r_k)$ such that $LCS(T, C\langle 1,k\rangle) = \max\{\sum_{i=1}^{k} (LCS(T[r_{i-1}+1, r_i], C_i))\}$, where $r_0 = 0$, $r_k = n$ and $C_i \in S$.*

Proof. We can expand $LCS(T, C\langle 1,k\rangle)$ based on Lemma 2 recursively until the base condition of $C\langle i,j\rangle$, "if $i = j$", is satisfied. Then we can compute the best LCS for each segment corresponding to r_i in T and each C_i, which will be the best sequence S_j in S, such that $LCS(T[r_{i-1}+1, r_i], S_j)$ is maximal. □

Hence, we can separate the mosaic sequence recursively and apply the dynamic programming approach for solving this problem. Let $L(l,p,q)$ denote the best LCS length of all possible l-mosaic sequences and $T[p,q]$. Our dynamic programming algorithm for the MLCS problem (FOR MOSAic) is presented in Algorithm Formosa1.

Theorem 4. *Algorithm Formosa1 solves the MLCS problem in $O(n^2 m|S| + n^3 \log k)$ time.*

Proof. Based on Lemmas 2 and 3, we can divide the problem into subproblems, and then merge small ones into large ones and the original problem finally. Algorithm Formosa1 computes each $L(l,p,q)$ bottom-up, which means all possible situations have been considered. Hence, $L(k,1,n)$ is the maximal LCS length. Step 1 takes $O(\log_2 k)$ time to convert k into its binary representation. Step 2 computes the LCS of each sequence in S and each substring of T. While fixing S_j and the p of $T[p,q]$, we can compute $LCS(T[p,q], S_j)$ in $O(mn)$ time for all $p \leqslant q \leqslant n$. There are $|S|$ sequences in S and n starting positions p in T. Hence, Step 2 requires $O(n^2 m|S|)$ time totally. In Step 3, it requires $O(n^3 \log k)$ to compute $L(2^i, p, q)$ for various (i,p,q) according to the best break point r. The time complexity of Step 4 is similar to that of Step 3; the number k can be represented as a binary string b with length $\lfloor \log_2 k \rfloor$ and we can compute $L(b[i,0], p, q)$ by assembling suffix string of b progressively. Step 5 is a simple return in constant time. In summary, the time complexity of Algorithm Formosa1 is $O(n^2 m|S| + n^3 \log k)$. □

Algorithm Formosa1

Input: A target sequence T, a source sequence set $S = \{S_1, S_2, \ldots, S_{|S|}\}$, and the mosaic number k.
Output: $LCS(T, C) = L(k,1,n)$, where $C = C_1 C_2 \ldots C_k$ is the best mosaic sequence and $C_i \in S$ for $1 \leqslant i \leqslant k$.
Step 1: Calculate the binary representation of k, denoted as $b = b_w b_{w-1} \ldots b_2 b_1 b_0$, where $w = \lfloor \log_2 k \rfloor$ and $b_i \in \{0,1\}$ for $0 \leqslant i \leqslant w$.
Step 2: Let $L(2^0, p, q) = \max_j\{LCS(T[p,q], S_j)\}$, where $1 \leqslant p \leqslant q \leqslant n$ and $1 \leqslant j \leqslant |S|$.
Step 3: Compute $L(2^i, p, q) = \max_r\{L(2^{i-1}, p, r) + L(2^{i-1}, r+1, q)\}$, where $1 \leqslant p \leqslant r \leqslant q \leqslant n$ and $1 \leqslant i \leqslant \lfloor \log_2 k \rfloor$.
Step 4: If $k \neq 2^w$, $L(b[i,0], p, q) = \max_r\{L(2^i \times b_i, p, r) + L(b[i-1,0], r+1, q)\}$, where $1 \leqslant p \leqslant r \leqslant q \leqslant n$, $1 \leqslant i \leqslant \lfloor \log_2 k \rfloor$, and $b[i,0]$ denotes the value of the binary string $b_i b_{i-1} \ldots b_0$.
Step 5: Return $L(k,1,n)$.

Algorithm Formosa1 computes all small mosaic sequences corresponding to every substring of T to assemble the final k-mosaic sequence. We now consider only the prefix string $T[1,q]$ to replace all substrings $T[p,q]$.

The *S-table* of two sequences T and X, denoted as $S^t(T,X)$, stores the LCS length of X and each suffix string $T[i,n]$ of T, $1 \leqslant i \leqslant n$ [7,6]. There exist efficient algorithms for finding $LCS(T[1,j], C_1C_2)$, $1 \leqslant j \leqslant n$, while $LCS(T[1,i], C_1)$ for $1 \leqslant i \leqslant n$ and the $S^t(T, C_2)$ are given. Here, we will invoke algorithms that can merge two preprocessed LCS's with $O(mn)$ preprocessing time [7,6] and $O(n)$ time for merging [1, 7]. In addition, algorithm SMAWK [1] is a merging algorithm with linear time but it is a recursive one. Landau and Ziv-Ukelson proposed a non-recursive merging algorithm for the edit distance problem with the same complexity [7]. The symbol $\oplus$ denotes the merging operation of a vector of $LCS(T[1,i], C_1)$ and an S-table. Summarizing the previous results, there is a lemma as follows [1,7,6].

Lemma 5. [1,7,6] *Given a sequence T and a composite sequence $C = C_1C_2$, $LCS(T[1,j], C_1)$, $1 \leqslant j \leqslant n$, and $S^t(T, C_2)$ can be obtained in $O(mn)$ time. If $LCS(T[1,j], C_1)$ and $S^t(T, C_2)$ are given, $LCS(T[1,j], C_1C_2)$ can be obtained by merging $LCS(T[1,j], C_1) \oplus S^t(T, C_2)$ in $O(n)$ time.*

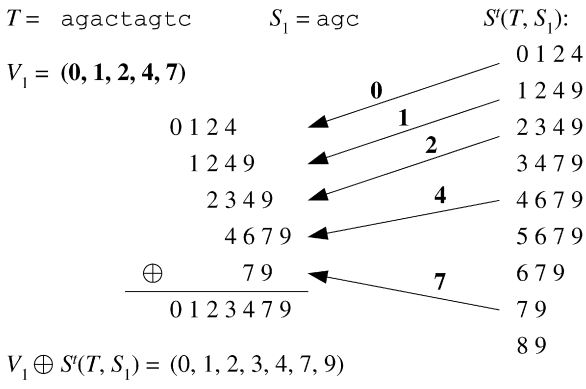

Fig. 1. The detail of calculating $V_1 \oplus S^t(T, S_1)$.

Let $L'(l, j)$ denote the best of $LCS(T[1, j], C\langle 1, l\rangle)$ among all l-mosaic sequences and $T[1, j]$, $1 \leqslant l \leqslant k$ and $1 \leqslant j \leqslant n$. We define a vector $V_l = (V_{l0}, V_{l1}, V_{l2}, \ldots, V_{l\lambda})$, where $V_{l0} = 0$, $\lambda = L'(l, n)$, and $V_{li} = \min\{j \mid L'(l, j) = i\}$, $1 \leqslant i \leqslant \lambda$. We use the function $minE()$ to get the minimal value in each position among several vectors. For example, $minE\{(5, 1, 6), (2, 3, 4, 1), (2, 2, 3)\} = (2, 1, 3, 1)$. Our improved algorithm for solving the MLCS problem by using the S-table is shown in Algorithm Formosa2.

For example, suppose $T = \texttt{agactagtc}$, and $S = \{S_1 = \texttt{agc}, S_2 = \texttt{act}, S_3 = \texttt{aatg}, S_4 = \texttt{ttcg}\}$. $V_0 \oplus S^t(T, S_i)$ are $(0, 1, 2, 4)$, $(0, 1, 4, 5)$, $(0, 1, 2, 5, 7)$, and $(0, 2, 7, 9)$ for $i = 1, 2, 3$, and 4, respectively. Hence, $V_1 = minE\{(0, 1, 2, 4), (0, 1, 4, 5), (0, 1, 2, 5, 7), (0, 2, 7, 9)\} = (0, 1, 2, 4, 7)$. Fig. 1 shows the detail of calculating $V_1 \oplus S^t(T, S_1)$, which is $(0, 1, 2, 3, 4, 7, 9)$, that reflects the best $LCS(T[1, j], C\langle 1, 1\rangle S_1)$, where $C\langle 1, 1\rangle \in S$. One can check that $LCS(T[1, j], \texttt{agc}\,\texttt{agc})$ are a, ag, aga, agac, agcag, and agcagc for $j = 1, 2, 3, 4, 7$, and 9, respectively. Then, with similar calculation, we get $V_2 = minE\{(0, 1, 2, 3, 4, 7, 9), (0, 1, 2, 3, 4, 5), (0, 1, 2, 3, 5, 7), (0, 1, 2, 4, 5, 7, 9)\} = (0, 1, 2, 3, 4, 5, 9)$. Finally, $V_3 = (0, 1, 2, 3, 4, 5, 6, 7, 9)$ and $V_4 = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)$ can be obtained.

Theorem 6. *Algorithm Formosa2 solves the MLCS problem in* $O(n(m + k)|S|)$ *time.*

Proof. Algorithm Formosa2 is based on Lemmas 5 and 2 with $k' = k - 1$ recursively. Each element in V_l keeps the best $LCS(T[1, i], C\langle 1, l\rangle)$ for $1 \leqslant i \leqslant n$ and its corresponding $C\langle 1, l\rangle$. And, $L'(k, n)$ is the best of $LCS(T[1, n], C\langle 1, k\rangle)$ among all possible k-mosaic sequences. In other words, $L'(k, n)$ is the MLCS length. Based on Lemma 5, we can obtain $S^t(T, S_i)$ while computing $LCS(T, S_i)$ for each $S_i \in S$ and hence Step 2 requires $O(nm|S|)$ time. Step 3 merges the vector V_l and the S-table of S_i, which each merge can be done in $O(n)$ time for $1 \leqslant l \leqslant k$ and $1 \leqslant i \leqslant |S|$, and each $minE()$ needs $O(n|S|)$ time. So, Step 3 requires $O(nk|S|)$ time. In summary, the time complexity of Algorithm Formosa2 is $O(n(m + k)|S|)$. □

Algorithm Formosa2

Input: A target sequence T, a source sequence set $S = \{S_1, S_2, \ldots, S_{|S|}\}$, and the mosaic number k.
Output: $LCS(T, C) = L'(k, n)$, where $C = C_1C_2 \ldots C_k$ is the best mosaic sequence and $C_i \in S$ for $1 \leqslant i \leqslant k$.
Step 1: Initialize $V_0 = \mathbf{0}$.
Step 2: Compute and store S-table $S^t(T, S_i)$ for each $S_i \in S$.
Step 3: $V_l = minE\{V_{l-1} \oplus S^t(T, S_i)\}$ for each $S_i \in S$ and $1 \leqslant l \leqslant k$.
Step 4: Return $L'(k, n)$.

After Algorithm Formosa2 finishes, it is easy to find out the MLCS sequence with the tracing back technique as follows.

Theorem 7. *If the k-mosaic sequence C and its corresponding break point vector* $\mathbf{r}$ *of T are given, the sequence of $LCS(T, C)$ can be obtained in* $O(mn)$ *time.*

Proof. Based on the break point vector $\mathbf{r}$, we can separate T into k segments, i.e., $T = T_1T_2 \ldots T_k$. Each segment T_i has to consider only its corresponding C_i. We can use simple dynamic programming formula to find out the sequence of $LCS(T_i, C_i)$ in $O(|T_i||C_i|)$ time. Thus, the time required for all k segment pairs of (T_i, C_i) is $O(|T|m) = O(mn)$. □

We note that the time complexity of tracing back the MLCS sequence is lower than both algorithms in this paper.

4. Conclusion

We define the MLCS problem for finding $LCS(T, C)$ of a mosaic sequence C, composed of repeatable k sequences in set S of source sequences. Based on the concept of break points in sequence T, we first propose a divide-and-conquer algorithm with $O(n^2m|S| + n^3 \log k)$ time for solving the MLCS problem, where n and m are the length of T and the maximal length of sequences in S, respectively. Furthermore, by applying the S-table, we propose an improved algorithm with $O(n(m + k)|S|)$ time for solving the problem.

There are still some problems worthy for further study. One of them is the chimeric LCS problem. The

chimeric sequence means a composite sequence composed of several independent (not repeatable) sequences in S. It is a more challenging combinatorial problem because of the permutation without repetition of the composite sequence. However, it may be less useful in biological applications than the mosaic one while k is large. In addition, one may study the more general mosaic alignment problems related to the MLCS problem, such as generalized score matrices, local alignment and gap penalty.

References

[1] A. Aggarwal, M.M. Klawe, S. Moran, P. Shor, R. Wilber, Geometric applications of a matrix-searching algorithm, Algorithmica 2 (1) (1987) 195–208.

[2] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, New York, 1997.

[3] D.S. Hirschberg, A linear space algorithm for computing maximal common subsequence, Communications of the ACM 18 (6) (1975) 341–343.

[4] G.A. Komatsoulis, M.S. Waterman, Chimeric alignment by dynamic programming: algorithm and biological uses, in: RECOMB '97: Proceedings of the First Annual International Conference on Computational Molecular Biology, New York, NY, USA, ACM Press, 1997, pp. 174–180.

[5] G.A. Komatsoulis, M.S. Waterman, A new computational method for detection of chimeric 16S rRNA artifacts generated by PCR amplification from mixed bacterial populations, Applied Environmental Microbiology 63 (6) (June 1997) 2338–2346.

[6] G.M. Landau, B. Schieber, M. Ziv-Ukelson, Sparse LCS common substring alignment, Information Processing Letters 88 (6) (2003) 259–270.

[7] G.M. Landau, M. Ziv-Ukelson, On the common substring alignment problem, Journal of Algorithms 41 (2) (2001) 338–354.

[8] D. Maier, The complexity of some problems on subsequences and supersequences, Journal of the ACM 25 (2) (1978) 322–336.

[9] C.B. Yang, R.C.T. Lee, Systolic algorithms for the longest common subsequence problem, Journal of the Chinese Institute of Engineers 10 (6) (1987) 691–699.