A Simple Linear Space Algorithm for Computing a Longest Common Increasing Subsequence

Danlin Cai, Daxin Zhu, Lei Wang, and Xiaodong Wang*

***Abstract*—This paper presents a linear space algorithm for finding a longest common increasing subsequence of two given input sequences in a very succinct way. The simple linear space algorithm based on a new formula can find a longest common increasing subsequence of sizes n and m respectively, in time $O(nm)$ using $O(n+m)$ space.**

***Index Terms*—longest common increasing subsequence, dynamic programming, time complexity**

I. Introduction

The study of the longest common increasing subsequence (LCIS) problem originated from two classical subsequence problems, the longest common subsequence (LCS) and the longest increasing subsequence (LIS). The classic algorithm to the LCS problem is the dynamic programming solution of Wagner et al. [9], [10], with $O(n^2)$ worst case running time. Masek et al. [7] improved this algorithm by using the "Four-Russians" technique to reduce its running time to $O(n^2/\log n)$ in the worst case. Since then, there has not been much improvement on the time complexity in terms of n found in the literature. There is also a rich history for the longest increasing subsequence problem, e.g., see [2], [3], [5], [12]. The problem can be solved in $O(n \log n)$ time for the input sequence of size n. The maximal length k of an increasing subsequence of a permutation of the set of integers $\{1, 2, \cdots, n\}$ can be computed in time $O(n \log \log k)$ in the RAM model [2].

The LCIS problem for input sequences X and Y consists of finding a subsequence Z of X and Y which is both an increasing sequence and a common subsequence of X and Y with maximal length among all common increasing subsequences of X and Y. Yang et al. [11] designed a dynamic programming algorithm that finds an LCIS of two input sequences of size n and m in $O(nm)$ time and space.

Manuscript received January 18, 2015; revised May 22, 2015.

This work was supported in part by Fujian Provincial Key Laboratory of Data-Intensive Computing and Fujian University Laboratory of Intelligent Computing and Information Processing.

Danlin Cai is with Quanzhou Normal University, Quanzhou, China.

Daxin Zhu is with Quanzhou Normal University, Quanzhou, China.(email:dex@qztc.edu.cn)

Lei Wang is with Facebook, 1 Hacker Way, Menlo Park, CA 94052, USA.

Xiaodong Wang is with Fujian University of Technology, Fuzhou, China.

*Corresponding author.

Subsequently to [11], several faster algorithms were obtained for the LCIS problem in special cases. If the length of the LCIS, l, is small, Kutz et al. [6] gave a faster algorithm which runs in $O(nl \log n)$ time. If $x_i = y_j$, then we say the ordered pair of position (i, j) is a match of the input sequences X and Y. If r, the total number of ordered pairs of positions at which the two input sequences match, is relatively small, Chan et al. [1] gave a faster algorithm which runs in $O(\min(r \log l, nl + r) \log \log n + n \log n)$ time where n is the length of each sequence. A first linear space algorithm was proposed by Sakai [8]. The time and space costs of the Sakai's algorithm are $O(mn)$ and $O(m + n)$ respectively. The space cost of the algorithm of Yang et al. was reduced from $O(mn)$ to $O(m + n)$ by a careful application of Hirschberg's divide-and-conquer method [4]. The space complexity of the algorithm of Kutz et al. [6] was also reduced from $O(nl)$ to $O(m)$ by using the same divide-and-conquer method of Hirschberg [4].

In this paper, we find a very simple linear space algorithm to solve the problem based on a novel recursive formula. In particular, the algorithm of Kutz et al. [6] has the same linear space complexity as our algorithm, although the time complexities of these algorithms are incomparable when the length of the LCIS of X and Y is unrestricted. Our new algorithm shows that the longest common increasing subsequence (LCIS) problem can be solved in a very similar approach to the standard Wagner algorithm for the longest common subsequence (LCS) problem, which is definitely not obvious. Still, it is a rather basic dynamic-programming algorithm, and essentially follows from just two basic observations:

(1) An LCS can be found by computing $LCS((x_1, ..., x_i), (y_1, ..., y_j))$, the length of the LCS of the length-i and length-j prefixes of X and Y, respectively, for all i, j. This does not seem to work directly for LCIS. Instead, we can compute $f(i, j)$, which is the length of the LCIS of the length-i and length-j prefixes of X and Y that ends on y_j. This additional restriction is crucial.

(2) By computing the entries in the appropriate order over i, j, there are simple update rules for computing $f(i, j)$ and an additional auxiliary quantity $\theta_{i,j}$.

The most fundamental idea of our algorithm using only linear space is as follows. The Hirschberg's divide-and-

conquer algorithm for the LCS problem consists of the following two steps. In the first step, two linear-length arrays containing information about a midpoint are constructed in quadratic time. In the second step, A midpoint, which divides the problem into two sub-problems, from information contained in the arrays is extracted in linear time. However, if the second step is applied naively to the LCIS problem, the midpoint cannot be extracted correctly in linear time. In order to overcome this situation, our algorithm takes a split algorithm such that the second step can be executed in quadratic time.

II. Definitions and Terminologies

In the whole paper we will use $X = x_1x_2\cdots x_n$ and $Y = y_1y_2\cdots y_m$ to denote the two input sequences of size n and m respectively, where each pair of elements in the sequences is comparable.

Some terminologies on the LCIS problem are usually referred to in the following way.

Definiton 1:

A **subsequence** of a sequence X is obtained by deleting zero or more characters from X (not necessarily contiguous). If Z is a subsequence of both sequences X and Y, then Z is called a **common subsequence** of X and Y. If $z_1 < z_2 < \cdots < z_k$, then $Z = z_1z_2\cdots z_k$ is called an **increasing sequence**. The **longest common increasing subsequence** of X and Y is defined as a common increasing subsequence whose length is the longest among all common increasing subsequences of the two given sequences.

Example.

Let $X = (3, 5, 1, 2, 7, 5, 7)$ and $Y = (3, 5, 2, 1, 5, 7)$. We have that $n = 7$ and $m = 6$. $Z = (3, 1, 2, 5)$ is a subsequence of X with corresponding index sequence $(1, 3, 4, 6)$.

The subsequence $(3, 5, 1)$ and $(3, 5, 7)$ are common subsequences of both X and Y, and the subsequence $(3, 5, 7)$ is an LCIS of X and Y.

Definiton 2:

For a sequence $X = x_1x_2\cdots x_n$, its substring $x_ix_{i+1}\cdots x_j, 1 \le i \le j \le n$, is denoted as $X_{i,j}$. When $i = 1$, $X_{1,j}$ is also denoted as X_j. The empty prefix of X is defined as X_0.

For each pair $(i, j), 0 \le i \le n, 0 \le j \le m$, the set of all LCISs of X_i and Y_j that **ends on** y_j is denoted by $LCIS(X_i, Y_j)$. The length of an LCIS in $LCIS(X_i, Y_j)$ is denoted as $f(i, j)$.

The length of a string S is denoted by $|S|$. The concatenation of two strings S_1 and S_2 is denoted by S_1S_2.

Definiton 3:

A match of the two sequences $X = x_1x_2\cdots x_n$ and $Y = y_1y_2\cdots y_m$ is defined as an ordered pair $(i, j), 1 \le i \le n, 1 \le j \le m$ such that $x_i = y_j$.

Definiton 4:

For each index $j, 1 \le j \le m$, the index set $\beta(j)$ is defined as follows:

$$\beta(j) = \{t \mid 1 \le t < j \ \texttt{and} \ y_t < y_j\} \tag{1}$$

III. A recursive formula

Similar to the $O(nm)$ solution of Wagner and Fischer for computing the length of an LCS, a standard dynamic programming algorithm can be built based on the following recurrence for the length $f(i, j)$ of an LCIS in $LCIS(X_i, Y_j)$, $1 \le i \le n, 1 \le j \le m$.

Theorem 1:

Let $X = x_1x_2\cdots x_n$ and $Y = y_1y_2\cdots y_m$ be two input sequences over an alphabet $\sum$ of size n and m respectively. For each $0 \le i \le n, 0 \le j \le m$, $f(i, j)$, the length of an LCIS of X_i and Y_j that ends on y_j, can be computed by the following dynamic programming formula.

$$f(i,j) = \begin{cases} 0, & \text{if } i = 0, \\ f(i-1,j), & \text{if } i > 0 \text{ and } x_i \neq y_j, \\ 1 + \max\limits_{t\in\beta(j)} f(i-1,t), & \text{if } i > 0 \text{ and } x_i = y_j. \end{cases}$$

Proof:

(1) The initial case is trivial.

(2) In the case of $x_i \neq y_j$, we have that $Z \in LCIS(X_i, Y_j)$ if and only if $Z \in LCIS(X_{i-1}, Y_j)$, and thus $LCIS(X_i, Y_j) = LCIS(X_{i-1}, Y_j)$. Therefore, $f(i, j) = f(i - 1, j)$.

(3) In the case of $x_i = y_j$, let $Z = z_1z_2\cdots z_k \in LCIS(X_i, Y_j)$ be an LCIS of X_i and Y_j that ends on y_j. In this case, we have that $f(i, j) = k$, and $z_1z_2\cdots z_{k-1}$ must be a common increasing subsequence of X_{i-1} and Y_t for some $1 \le t < j$, and $z_{k-1} = y_t < y_j$. It follows that $k - 1 \le LCIS(X_{i-1}, Y_t)$, and thus

$$f(i, j) \le 1 + \max_{t\in\beta(j)} f(i - 1, t) \tag{2}$$

On the other hand, let $Z = z_1z_2\cdots z_k \in LCIS(X_{i-1}, Y_t)$ for some $1 \le t < j$ such that $z_k = y_t < y_j$, then Zy_j, the concatenation of Z and y_j, must be a common increasing subsequence of X_i and Y_j ending on y_j. This means, $k+1 \le f(i, j)$, and thus $f(i - 1, t) + 1 \le f(i, j)$. It follows that

$$f(i, j) \ge 1 + \max_{t\in\beta(j)} f(i - 1, t) \tag{3}$$

Combining (2) and (3), we have $f(i, j) = 1 + \max\limits_{t\in\beta(j)} f(i - 1, t)$.

The proof is complete. ■

Based on Theorem 1, the length of LCISs for the given input sequences $X = x_1x_2\cdots x_n$ and $Y = y_1y_2\cdots y_m$

of size n and m respectively, can be computed in $O(nm)$ time and $O(nm)$ space by a standard dynamic programming algorithm.

Algorithm 1: LCIS

Input: X, Y
Output: $f(i,j)$, the length of LCIS of X_i and Y_j ending on y_j
for j=1 **to** m **do** $f(0,j) \leftarrow 0$;
for i=1 **to** n **do**
  $\theta \leftarrow 0$; // initialize θ to $\max_{t\in\beta(1)} f(i-1,t)$
  for j=1 **to** m **do**
    if $x_i \neq y_j$ **then** $f(i,j) \leftarrow f(i-1,j)$ **else** $f(i,j) \leftarrow \theta+1$; // compute $f(i,j)$
    if $x_i > y_j$ **then** $\theta \leftarrow \max\{\theta, f(i-1,j)\}$; // update θ to $\max_{t\in\beta(j+1)} f(i-1,t)$
  end
end
return $\max_{1\le j\le m} f(n,j)$

In the for loop of the above algorithm at iteration (i,j), if $x_i > y_j$, then θ is updated to $\max_{t\in\beta(j+1)} f(i-1,t)$, and thus $\theta = \max\limits_{t\in\beta(j)} f(i-1,t)$ when $x_i = y_j$. The correctness is therefore insured by Theorem 1. It is clear that the time and space complexities of the algorithm are both $O(nm)$.

Based on the formula (**??**), we can reduce the space cost of the algorithm $LCIS$ to $\min\{n,m\}+1$. When computing a particular row of the dynamic programming table, no rows before the previous row are required. Only two rows have to be kept in memory at a time, and thus we need only $\min\{n,m\}+1$ entries to compute the table.

IV. A LINEAR SPACE ALGORITHM

A. The description of the algorithm

If a longest common increasing subsequence has to be constructed, not just its length, then the information provided by L is not enough for this purpose. The linear space algorithm proposed in this section is also based on Hirschberg's divide-and-conquer method of solving the LCS problem in linear space [4]. In order to use the divide-and-conquer method, we need to extend the definition of LCIS to a more general definition of bounded LCIS as follows.

Definiton 5:

Let l and u be the given lower and upper bounds of a sequence, respectively. If $Z = z_1 z_2 \cdots z_k$ is a common increasing subsequence of X and Y satisfying $l < z_1 < z_2 < \cdots < z_k < u$, then Z is called a **bounded common increasing subsequence** of X and Y.

For the two substrings X_{i_0,i_1} and Y_{j_0,j_1}, $1 \le i_0 \le i_1 \le n, 1 \le j_0 \le j_1 \le m$, and the lower bound l and the upper bound u, if $i_0 \le i \le i_1$ and $j_0 \le j \le j_1$, then the set of all bounded LCISs of $X_{i_0,i}$ and $Y_{j_0,j}$ that ends on y_j is denoted by $LCIS(i_0,i_1,j_0,j_1,i,j,l,u)$. The length of a bounded LCIS in $LCIS(i_0,i_1,j_0,j_1,i,j,l,u)$ is denoted as $g(i_0,i_1,j_0,j_1,i,j,l,u)$. Similarly, the set of all bounded LCISs of X_{i,i_1} and Y_{j,j_1} that begins with y_j is denoted by $LCISR(i_0,i_1,j_0,j_1,i,j,l,u)$. The length of a bounded LCIS in $LCISR(i_0,i_1,j_0,j_1,i,j,l,u)$ is denoted as $h(i_0,i_1,j_0,j_1,i,j,l,u)$.

In the special case of $i_0 = 1, i_1 = n$ and $j_0 = 1, j_1 = m$, $g(1,n,1,m,i,j,l,u)$ is denoted as $f(i,j,l,u)$ compared to $f(i,j)$, the length of the unbounded LCIS of X_i and Y_j that ends on y_j.

It is clear that $f(i,j) = g(1,n,1,m,i,j,-\infty,\infty) = f(i,j,-\infty,\infty), 1 \le i \le n, 1 \le j \le m$. The algorithm to compute $g(i_0,i_1,j_0,j_1,i,j,l,u)$ is essentially the same as the algorithm to compute $f(i,j)$. A linear space algorithm to compute $g(i_0,i_1,j_0,j_1,i,j,l,u)$ can be described as follows.

Algorithm 2: $g(i_0,i_1,j_0,j_1,l,u)$

Input: i_0,i_1,j_0,j_1,l,u
Output: L
for $i = i_0$ *to* i_1 **do**
  if $x_i > l$ **and** $x_i < u$ **then**
    $L(0) \leftarrow 0$;
    for $j = j_0$ *to* j_1 **do**
      if $x_i > y_j$ **and** $L(j) > L(0)$ **then** $L(0) \leftarrow L(j)$;
      if $x_i = y_j$ **then** $L(j) \leftarrow L(0)+1$;
    end
  end
end
return L

In the for loop of the above algorithm at iteration (i,j), if $x_i > y_j$, then $L(j) = g(i_0,i_1,j_0,j_1,i-1,j,l,u)$, and it is compared with $L(0)$, and thus $L(0) = \max\limits_{t\in\beta(j)} g(i_0,i_1,j_0,j_1,i-1,t,l,u)$ when $x_i = y_j$. Since only matched pairs are checked in the algorithm, when the match (i,j) is checked, we have $x_i = y_j$ and $l < x_i < u$, and thus $l < y_j < u$. Therefore, there is no need to check $l < y_j < u$ explicitly in the algorithm.

At the end of the algorithm, the value of $g(i_0,i_1,j_0,j_1,i_1,j,l,u)$ is stored in $L(j)$.

It is clear that to compute an LCIS in $LCISR(i_0,i_1,j_0,j_1,i,j,l,u)$ is equivalent to compute a longest decreasing subsequence for the two reversed strings $\widehat{X}_{i,i_1}$ and $\widehat{Y}_{j,j_1}$ that ends on y_j with lower and upper bounds l and u, where $\widehat{X}_{i,i_1} = x_{i_1} x_{i_1-1}, \cdots, x_i$

and $\widehat{Y}_{j,j_1} = y_{j_1} y_{j_1-1}, \cdots, y_j$. Therefore, the algorithm to compute $h(i_0, i_1, j_0, j_1, i, j, l, u)$ is also almost the same as the algorithm to compute $f(i, j)$. A linear space algorithm to compute $h(i_0, i_1, j_0, j_1, i, j, l, u)$ can be described similarly.

Let l, u be fixed integers. For simplicity, denote

$$g(i,j) = \begin{cases} g(1,i,1,m,i,j,l,u), & \text{if } 1 \le i \le n, 1 \le j \le m, \\ 0, & \text{otherwise.} \end{cases} \quad (4)$$

$$h(i,j) = \begin{cases} h(i,n,1,m,i,j,l,u), & \text{if } 1 \le i \le n, 1 \le j \le m, \\ 0, & \text{otherwise.} \end{cases} \quad (5)$$

and

$$M(i,l,u) = \max_{0 \le j \le m} \max_{j<t} \{g(i,j) + h(i+1,t) \mid y_j < y_t\} \quad (6)$$

Like the function $M(i)$ defined in [4], the function $M(i, l, u)$ defined above is the maximal length of the two merged LCIS for the subsequences of X spitted at position i and Y.

Theorem 2: For $0 \le i \le n$,

$$M(i,l,u) = \max_{0 \le j \le m} f(n,j,l,u)$$

Proof:

Let $M(i, l, u) = g(i, j) + h(i + 1, t)$ for some j, t and $j < t, y_j < y_t$. Let $Z(i, j) \in LCIS(1, i, 1, m, i, j, l, u)$ and $Z'(i, j) \in LCISR(i + 1, n, t, m, i + 1, t, l, u)$. Then, $C = Z(i, j)Z'(i, j)$, the concatenation of the two LCISs, is a common increasing subsequence of X and Y of length $M(i, l, u)$, since $j < t$ and $y_j < y_t$. Therefore, $M(i, l, u) \le \max_{0 \le j \le m} f(n, j, l, u)$.

On the other hand, let $\max_{0 \le j \le m} f(n, j, l, u) = f(n, j', l, u) = g(n, j')$ for some j' such that $Z(n, j') \in LCIS(1, n, 1, m, n, j', l, u)$. For $0 \le i \le n$, $Z(n, j')$ can be written as $Z(n, j') = S_1 S_2$, where S_1 is an increasing subsequence of $X_{1,i}$ and S_2 is an increasing subsequence of $X_{i+1,n}$. Then there must be s and t such that $s < t, y_s < y_t$, S_1 is an increasing subsequence of $Y_{1,s}$, and S_2 is an increasing subsequence of $Y_{t,m}$. By definition of g and h, $|S_1| \le g(i, s)$ and $|S_2| \le h(i + 1, t)$. Thus $g(n, j') = |S_1| + |S_2| \le g(i, j) + h(i + 1, t) = M(i, l, u)$, i.e., $\max_{0 \le j \le m} f(n, j, l, u) \le M(i, l, u)$. ■

It is clear that for $0 \le i \le n$, $M(i, -\infty, \infty) = \max_{0 \le j \le m} f(n, j, -\infty, \infty) = \max_{0 \le j \le m} f(n, j)$. This is the length of the LCIS of X and Y.

We now can use the above theorem recursively to design a divide-and-conquer algorithm to find an LCIS of X and Y as follows.

Algorithm 3: $D\&C(i_0, i_1, j_0, j_1, l, u)$

Input: i_0, i_1, j_0, j_1, l, u

Output: A bounded LCIS of $X_{i_0 i_1}$ and $Y_{j_0 j_1}$

```
p ← i_1 − i_0 + 1;
if p > 1 then
    i ← i_0 + ⌊p/2⌋ − 1;
    L_1 ← g(i_0, i, j_0, j_1, l, u);
    L_2 ← h(i + 1, i_1, j_0, j_1, l, u);
    (s, t) ← split(j_0, j_1, L_1, L_2);
    if s > j_0 then D&C(i_0, i − 1, j_0, s − 1, l, y_s);
    if s > 0 then print y_s;
    if t > 0 then print y_t;
    if t > 0 and t < j_1 then
    D&C(i + 2, i_1, t + 1, j_1, y_t, u);
end
else if p = 1 and x_{i_0} > l and x_{i_0} < u then
    for j = j_0 to j_1 do
        if x_{i_0} = y_j then print x_{i_0}; return;
    end
end
```

In the above algorithm, the sub-algorithm $split(j_0, j_1, L_1, L_2)$ is used to find the left and right split points s and t by using Theorem 2, where L_1 and L_2 are computed by g (Algorithm 2) and h respectively.

Algorithm 4: $split(j_0, j_1, L_1, L_2)$

Input: j_0, j_1, L_1, L_2

Output: s, t

```
s, t, sum ← 0;
for j = j_0 to j_1 do if L_2(j) > sum then
sum ← L_2(j), t ← j;
for j = j_0 to j_1 do
    if L_1(j) > sum then sum ← L_1(j), s ← j, t ← 0;
    for k = j + 1 to j_1 do
        if y_k > y_j and L_1(j) + L_2(k) > sum then
        sum ← L_1(j) + L_2(k), s ← j, t ← k;
    end
end
return (s, t)
```

With the returned values of s and t, the problem can then be divided into two smaller subproblems of finding LCISs in $LCIS(i_0, i - 1, j_0, s - 1, l, y_s)$ and $LCIS(i + 2, i_1, t + 1, j_1, y_t, u)$. In the case of $s = 0$ is returned, the solution is contained in L_2, and thus the $D\&C$ call for the first half of the subproblem can stop. Similarly, if $t = 0$ is returned, the solution is contained in L_1, and thus the $D\&C$ call for the second half of the subproblem can stop.

B. Correctness of the algorithm

We now prove that for the given l and u, if the above algorithm is applied to the given sequences X and Y, $D\&C(1,n,1,m,l,u)$ will produce a bounded LCIS of X and Y with lower and upper bounds l and u. The claim can be proved by induction on n and m, the sizes of the input sequence X and Y respectively. In the case of $n = 1$ and any $m > 0$, x_1 is a bounded LCIS if and only if there exists an index $j, 1 \leq j \leq m$ such that $x_1 = y_j$ and $l < x_1 < u$. This is verified in lines 11-15 of the algorithm. Thus, the claim is true for the initial cases of $n = 1$ and any $m > 0$.

Suppose the claim is true when the size of the input sequence X is less than n. We now prove that when the size of the input sequence X is n, the claim is also true. In this case, X is divided into two subsequences $X_{1,\lfloor n/2 \rfloor}$ and $X_{\lfloor n/2 \rfloor+1,n}$. Then, in line 4-5 of the algorithm, the length of a bounded LCIS in $LCIS(1,\lfloor n/2 \rfloor,1,m,\lfloor n/2 \rfloor,j,l,u)$ is computed by $g(1,\lfloor n/2 \rfloor,1,m,l,u,L_1)$ and the result is stored in $L_1(j), 1 \leq j \leq m$. The length of a bounded LCIS in $LCISR(\lfloor n/2 \rfloor+1,n,1,m,\lfloor n/2 \rfloor+1,j,l,u)$ is also computed by $h(\lfloor n/2 \rfloor+1,n,1,m,l,u,L_2)$ and the result is stored in $L_2(j), 1 \leq j \leq m$. $M(\lfloor n/2 \rfloor,l,u) = \max\limits_{0 \leq j \leq m} \max\limits_{j<t} \{g(\lfloor n/2 \rfloor,j) + h(\lfloor n/2 \rfloor+1,t) \mid y_j < y_t\}$ is then computed by the algorithm $split(s,t,1,m,L_1,L_2)$ in line 6, using the results in L_1 and L_2. The left and right split points s and t are found such that

In lines 7 and 9 of the algorithm, the LCISs $Z_1 \in LCIS(1,\lfloor n/2 \rfloor-1,1,s-1,l,y_s)$ and $Z_2 \in LCIS(\lfloor n/2 \rfloor+2,n,t+1,m,y_t,u)$ are found recursively, where $y_s < u$ is an upper bound of Z_1 and $y_t > l$ is a lower bound of Z_2. It follows from (7) and by induction that

$$\begin{cases} Z_1 y_s \in LCIS(1,\lfloor n/2 \rfloor,1,s,l,u) \\ y_t Z_2 \in LCIS(\lfloor n/2 \rfloor+1,n,t,m,l,u) \\ |Z_1 y_s| + |y_t Z_2| = M(\lfloor n/2 \rfloor,l,u) \\ y_s < y_t \end{cases} \tag{8}$$

Thus, $Z = (Z_1 y_s)(y_t Z_2)$ is a common increasing subsequence of X and Y with lower and upper bounds l and u. It follows from (8) and Theorem 2 that $|Z| = M(\lfloor n/2 \rfloor,l,u) = \max\limits_{0 \leq j \leq m} f(n,j,l,u)$. Therefore, Z, the common increasing subsequence of X and Y produced by the algorithm, is an LCIS of X and Y with lower and upper bounds l and u.

C. Time analysis of the algorithm

We have proved that a call $D\&C(1,n,1,m,0,\infty)$ of the algorithm produces an LCIS of X and Y. Let the time cost of the algorithm be $T(n,m)$ if the sizes of the input sequences are n and m respectively. The problem is divided into two smaller subproblems of finding LCISs in $LCIS(1,\lfloor n/2 \rfloor-1,1,s-1,l,y_s)$ and $LCIS(\lfloor n/2 \rfloor+2,n,t+1,m,y_t,u)$ by a call of $split(s,t,1,m,L_1,L_2)$ and two calls of $g(1,\lfloor n/2 \rfloor,1,m,l,u,L_1)$ and $h(\lfloor n/2 \rfloor+1,n,1,m,l,u,L_2)$. It is clear that the time costs of g and h are both $O(nm)$. Without loss of generality, we can assume that $m \leq n$, otherwise we can swap the roles of X and Y, and thus the time cost of $split$ is $O(m^2) = O(nm)$. Thus $T(n,m)$ satisfies the following equation.

Where, $0 \leq s < t \leq m$ and thus $s-1+m-t = m-1+s-t < m$. It can be proved by induction that (9) has a solution $T(n,m) = O(nm)$. The claim is obviously true for $T(1,m)$. Assume $T(n,m)$ is bounded by $c_1 \cdot nm$, and the $O(nm)$ term in (9) is bounded by $c_2 \cdot nm$. It follows from (9) that $T(\lfloor n/2 \rfloor-1,s-1)+T(n-\lfloor n/2 \rfloor-1,m-t)+O(nm)$ is bounded by

$$\begin{aligned} & c_1 \cdot ((\lfloor n/2 \rfloor-1)(s-1)+(n-\lfloor n/2 \rfloor-1)(m-t)) \\ & + c_2 \cdot nm \leq c_1 \cdot n/2(s-1+m-t)+c_2 \cdot nm \\ & \leq c_1 \cdot nm/2 + c_2 \cdot nm \\ & = (c_1/2+c_2) \cdot nm \end{aligned}$$

To be consistent with the assumption on the time bound of $T(n,m)$, we must have $c_1/2 + c_2 \leq c_1$, which is realizable by letting $c_1 \geq 2c_2$. It follows from (9) and by induction on n that $T(n,m) \leq c_1 \cdot nm$.

D. Space analysis of the algorithm

We assume that the input sequences X and Y are in common storage using $O(n+m)$ space. In the execution of the algorithm $D\&C$, the temporary arrays L_1 and L_2 are used in the execution of the algorithms g and h. It is clear that $|L_1| \leq m$ and $|L_2| \leq m$. It is seen that the execution of the algorithm $D\&C$ uses $O(1)$ temporary space, and the recursive calls to $D\&C$ are exclusive. There are at most $2n-1$ calls to the algorithm $D\&C$ (proved in [4]), and thus the space cost of the algorithm $D\&C$ is proportional to n and m, i.e. $O(n+m)$.

V. Concluding remarks

We have reformulated the problem of computing a longest common increasing subsequence of the two given input sequences X and Y of size n and m respectively. An extremely simple linear space algorithm based on the new formula can find a longest common increasing subsequence of X and Y in time $O(nm)$ using $O(n+m)$ space. Our new algorithm is much simpler than the $O(nm)$ time algorithm in [11], and the linear space algorithm in [8]. The time complexity of the new algorithm may be improved further.

$$\begin{cases} M(\lfloor n/2 \rfloor, l, u) = g(1, \lfloor n/2 \rfloor, 1, m, \lfloor n/2 \rfloor, s, l, u) + h(\lfloor n/2 \rfloor + 1, n, 1, m, n, t, l, u) \\ s < t \\ y_s < y_t \end{cases} \tag{7}$$

$$T(n,m) = \begin{cases} T(\lfloor n/2 \rfloor - 1, s-1) + T(n - \lfloor n/2 \rfloor - 1, m-t) + O(nm) & \text{if } n > 1, \\ O(m), & \text{if } n = 1. \end{cases} \tag{9}$$

REFERENCES

[1] W. Chan, Y. Zhang, S.P.Y. Fung, D. Ye, and H. Zhu, Efficient Algorithms for Finding a Longest Common Increasing Subsequence, *Journal of Combinatorial Optimization*, 13, 2007, pp. 277-288.

[2] M. Crochemore, E. Porat, Fast computation of a longest increasing subsequence and application,*Information and Computation* 208, 2010, pp. 1054-1059.

[3] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, Cambridge, UK, 1997.

[4] D.S. Hirschberg, A linear space algorithm for computing maximal common subsequences, *Commun. ACM* 18(6), 1975, pp. 341-343.

[5] J.W. Hunt, T.G. Szymanski, A fast algorithm for computing longest subsequences, *Commun. ACM* 20(5), 1977, pp. 350-353.

[6] M. Kutz, G.S. Brodal, K. Kaligosi, I. Katriel, Faster algorithms for computing longest common increasing subsequences, *Journal of Discrete Algorithms*, 9, 2011, pp.314-325.

[7] W.J. Masek and M.S. Paterson, A faster algorithm computing string edit distances, *J. Comput. System Sci.* 20, 1980, pp. 18-31.

[8] Y. Sakai, A linear space algorithm for computing a longest common increasing subsequence, *Information Processing Letters* 99, 2006, pp. 203-207.

[9] R.A. Wagner and M.J. Fischer, The string-to-string correction problem, *J. ACM* 21(1), 1974, pp.168-173.

[10] J. Yan, M. Li , and J. Xu , An Adaptive Strategy Applied to Memetic Algorithms, *IAENG International Journal of Computer Science*, vol. 42, no.2, 2015, pp73-84.

[11] I.H. Yang, C.P. Huang, K.M. Chao, A fast algorithm for computing a longest common increasing subsequence, *Information Processing Letters* 93 (5), 2005, pp. 249-253.

[12] D. Zhu , L. Wang , J. Tian, and X. Wang , A Simple Polynomial Time Algorithm for the Generalized LCS Problem with Multiple Substring Exclusive Constraints, *IAENG International Journal of Computer Science*, vol. 42, no.3, 2015, pp214-220.