The Longest Common Subsequence Problem for Arc-Annotated Sequences

Tao Jiang[1,4] *, Guo-Hui Lin[2,4] **, Bin Ma[2] ***, and Kaizhong Zhang[3] †

[1] Department of Computer Science, University of California, Riverside, CA 92521.
[2] Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada.
[3] Department of Computer Science, University of Western Ontario, London, Ontario N6A 5B7, Canada.
[4] Department of Computing and Software, McMaster University, Hamilton, Ontario L8S 4L7, Canada.

Abstract. Arc-annotated sequences are useful in representing the structural information of RNA and protein sequences. Recently, the longest arc-preserving common subsequence problem has been introduced in [6, 7] as a framework for studying the similarity of arc-annotated sequences. In this paper, we consider arc-annotated sequences with various arc structures and present some new algorithmic and complexity results on the longest arc-preserving common subsequence problem. Some of our results answer an open question in [6, 7] and some others improve the hardness results in [6, 7].

Keywords: sequence annotation, longest common subsequence, approximation algorithm, maximum independent set, MAX SNP-hard, dynamic programming.

1 Introduction

Given two sequences S and T over some fixed *alphabet* Σ, the sequence T is a *subsequence* of S if T can be obtained from S by deleting some letters from S. Notice that the order of the remaining letters of S bases must be preserved. The *length* of a sequence S is the number of letters in it and is denoted as $|S|$. Given two sequences S_1 and S_2 (over some fixed *alphabet* Σ), the classic *longest common subsequence (LCS)* problem asks for a longest sequence T that is a

* Supported in part by NSERC Research Grant OGP0046613, a CITO grant, and a UCR startup grant. Email: jiang@cs.ucr.edu.

** Supported in part by NSERC Research Grant OGP0046613 and a CITO grant. Email: ghlin@math.uwaterloo.ca.

*** Supported in part by NSERC Research Grant OGP0046506 and a CGAT grant. Email: b3ma@math.uwaterloo.ca.

† Email: kzhang@csd.uwo.ca. Supported in part by NSERC Research Grant OGP0046373.

subsequence of both S_1 and S_2. Suppose $|S_1| = n$ and $|S_2| = m$, the longest common subsequence of S_1 and S_2 can be found by dynamic programming in time $O(nm)$ [9, 13]. For simplicity, we use $S[i]$ to denote the ith letter in sequence S, and $S[i, j]$ to denote the substring of S consisting of the ith letter through the jth letter.

For any sequence S, an *arc annotation* (or *arc set*) P of S is a set of *unordered* pairs of positions in S. Each pair $(i, j) \in P$, where $1 \leq i < j \leq |S|$, is said to *connect* the two letters at positions i and j and is called an *arc* between the two letters. Such a pair (S, P) of sequence and arc annotation will be referred to as an *arc-annotated* sequence [6]. Observe that a (plain) sequence without any arc annotation can be viewed as an arc-annotated sequence with an empty arc set.

Arc-annotated sequences are useful in describing the secondary and tertiary structures of RNA and protein sequences [1, 4, 6, 8, 10, 12, 14]. For example, one may use arcs to represent bonds between nucleotides in an RNA sequence or contact forces between amino acids in a protein sequence. Therefore, the problem of comparing arc-annotated sequences has applications in the structural comparison of RNA and protein sequences and has received much recent attention in the literature [1, 6, 8, 10, 14]. In this paper, we follow the LCS approach proposed in [6] and study the LCS problem for annotated sequences.

1.1 The Arc-Annotated LCS Problem and Its Restricted Versions

Given two annotated sequences S_1 and S_2 with arc sets P_1 and P_2 respectively, a common subsequence T of S_1 and S_2 induces a bijective mapping from a subset of $\{1, \ldots, |S_1|\}$ to a subset of $\{1, \ldots, |S_2|\}$. The common subsequence T is *arc-preserving* if the arcs induced by the mapping are preserved, *i.e.*, for any (i_1, j_1) and (i_2, j_2) in the mapping,

$$(i_1, i_2) \in P_1 \Longleftrightarrow (j_1, j_2) \in P_2.$$

The *longest arc-preserving common subsequence* problem is to find a longest common subsequence of S_1 and S_2 that is arc-preserving (with respect to the given arc sets P_1 and P_2) [6].

It is shown in [6, 7] that the (general) longest arc-preserving common subsequence problem is NP-hard, if the arc annotations are unrestricted. Since in the practice of RNA and protein sequence comparison arc sets are likely to satisfy some constraints (*e.g.*, bond arcs do not cross in the case of RNA sequences), it is of interest to consider various restrictions on arc structures. We consider the following four natural restrictions on a arc set P which are first discussed in [6]:

1. no sharing of endpoints:
   $\forall (i_1, i_2), (i_3, i_4) \in P$, $i_1 \neq i_4$, $i_2 \neq i_3$, and $i_1 = i_3 \Longleftrightarrow i_2 = i_4$.
2. no crossing:
   $\forall (i_1, i_2), (i_3, i_4) \in P$, $i_1 \in [i_3, i_4] \Longleftrightarrow i_2 \in [i_3, i_4]$.
3. no nesting:
   $\forall (i_1, i_2), (i_3, i_4) \in P$, $i_1 \leq i_3 \Longleftrightarrow i_2 \leq i_3$.

4. no arcs:
   $P = \emptyset$.

These restrictions are used progressively and inclusively to produce five distinct *levels* of permitted arc structures for the longest arc-preserving common subsequence problem:

- *unlimited* — no restrictions;
- *crossing* — restriction 1;
- *nested* — restrictions 1 and 2;
- *chain* — restrictions 1, 2 and 3;
- *plain* — restriction 4.

1.2 Summary of Results

In the following, we will use the notation LCS(level-1, level-2) to represent the longest arc-preserving common subsequence problem where the arc structure of sequence S_1 is of level level-1 and the arc structure of sequence S_2 is of level level-2. Without loss of generality, we always assume that level-1 is at the same level or higher than level-2. In other words, we assume that the arc structure of sequence S_1 is at least as complex as that of sequence S_2.

For each position of a sequence, the number of arcs crossing it (including the arcs connecting to this position) is the *arc-cutwidth* of the sequence at this position [6]. The *cutwidth* of the sequence is defined to be the maximum arc-cutwidth over all positions. Given two arc-annotated sequences (S_1, P_1) and (S_2, P_2), the maximum of their cutwidths is defined as the cutwidth of the longest arc-preserving common subsequence problem concerning (S_1, P_1) and (S_2, P_2). Some of our results are concerned with sequences with bounded cutwidth.

The following table summarizes the algorithmic and complexity results on the longest arc-preserving common subsequence problem obtained in [6, 7] and this paper. Denote $|S_1| = n$ and $|S_2| = m$.

<table>
<tr><th></th><th>plain</th><th>chain</th><th>nested</th><th>crossing</th><th>unlimited</th></tr>
<tr><td>unlimited</td><td colspan="5">NP-hard [6, 7]</td></tr>
<tr><td></td><td colspan="5">not approximable within ratio n^ϵ, $\epsilon \in (0, \frac{1}{4})$</td></tr>
<tr><td>crossing</td><td colspan="4">NP-hard [6, 7]</td><td></td></tr>
<tr><td></td><td colspan="4">MAX SNP-hard</td><td></td></tr>
<tr><td></td><td colspan="4">admits a 2-approximation</td><td></td></tr>
<tr><td>nested</td><td colspan="2">$O(nm^3)$</td><td>complexity open</td><td></td><td></td></tr>
<tr><td></td><td colspan="2"></td><td>admits a 2-approximation</td><td></td><td></td></tr>
<tr><td>chain</td><td colspan="2">$O(nm)$ [6, 7]</td><td></td><td></td><td></td></tr>
<tr><td>plain</td><td>$O(nm)$ [9, 13]</td><td></td><td></td><td></td><td></td></tr>
</table>

Fig. 1. Results on LCS(level-1, level-2).

Our results on problems LCS(nested, chain) and LCS(nested, plain) in fact answer an open question in [6, 7].

The rest of the paper is organized as follows. In Section 2, we design the 2-approximation algorithm for problem LCS(crossing, crossing), which implies that problems LCS(crossing, nested), LCS(crossing, chain), LCS(crossing, plain), and LCS(nested, nested) all admit 2-approximation algorithms. In Section 3, we show that problem LCS(crossing, plain) is MAX SNP-hard. This implies that problems LCS(crossing, chain), LCS(crossing, nested) and LCS(crossing, crossing) are also MAX SNP-hard. This excludes the possibility for any of these problems to have a polynomial time approximation scheme (PTAS). In Section 4, we first design a dynamic programming algorithm to solve problem LCS(nested, plain) and then extend it to problem LCS(nested, chain) and to problem LCS(crossing, nested) where the first sequence has a bounded cutwidth. Some concluding remarks are given in Section 5.

2 Approximation Algorithms

The NP-hardness of problem LCS(crossing, plain) [6, 7] implies that problem LCS(level-1, level-2) is NP-hard whenever an input arc-annotated sequence has an unlimited or crossing arc structure. In the following, we give a 2-approximation algorithm for problem LCS(crossing, crossing), which also implies 2-approximation algorithms for problem LCS(crossing, nested), problem LCS(crossing, chain), and problem LCS(crossing, plain).

Let (S_1, P_1) and (S_2, P_2) be a pair of arc-annotated sequences. The algorithm begins with the classical dynamic programming for the classic LCS problem for the two plain sequences S_1 and S_2. Let Y denote the subsequence achieved and M denote the mapping between positions in S_1 and S_2 induced by Y. Assume that $|Y| = L$ and $M = \{(i_1, j_1), \ldots, (i_L, j_L)\}$.

We now construct a graph G_M, where each match (i_k, j_k) represents a vertex and two vertices (i_k, j_k) and (i_l, j_l) are connected by an edge if and only if either $(i_k, i_l) \in P_1$ or $(j_k, j_l) \in P_2$, but not both. Notice that the maximum degree of any vertex in G_M is at most 2. Consider for a connected component of G_M. If it contains more than one vertices, then it must be either a simple path or a simple cycle. It follows that we may delete at most half of the vertices in the component to make each of the remaining vertices isolated. Repeating this deletion procedure for all connected components containing more than one vertices until we get a graph G'_M, in which every component is a singleton. Let M' denote the subset of matches corresponding to those remaining vertices in G'_M. It follows that $|M'| \geq |M|/2$. Clearly, the subsequence inducing M' (which is obtained by deleting those letters corresponding to the deleted matches), denoted by Y', is an arc-preserving common subsequence. The high level description of the algorithm, called Algorithm $\mathcal{MLCS}$, is depicted in Figure 2.

Since $|Y'| \geq |Y|/2$ and $|Y|$ is an upper bound for the length of a longest arc-preserving common subsequence for (S_1, P_1) and (S_2, P_2), the above algorithm

Algorithm $\mathcal{MLCS}$
Input: An instance of problem LCS(crossing, crossing): (S_1, P_1) and (S_2, P_2). Output: an arc-preserving common subsequence.
1. Compute an LCS Y for (S_1, S_2) using dynamic programming. Let M denote the mapping induced by Y. 2. Construct the graph G_M for M. 3. **For** each connected component C containing more than one vertices 3.1 Delete (at most) half of the vertices therein to isolate the remaining vertices. 4. Let G'_M be the final graph and M' the set of matches corresponding to the vertices in G'_M. 5. Construct Y', the subsequence corresponding to mapping M'. 6. Output Y' as the solution subsequence.

Fig. 2. The algorithm $\mathcal{MLCS}$.

is a 2-approximation for LCS(crossing, crossing). Hence, we have the following theorem.

Theorem 1. *Problem* LCS(crossing, crossing) *has a* 2*-approximation algorithm with time complexity* $O(nm)$.

The following corollary is straightforward.

Corollary 1. *Problem* LCS(crossing, nested)*, problem* LCS(crossing, chain)*, and problem* LCS(crossing, plain) *have* 2*-approximation algorithms running in* $O(nm)$ *time.*

3 Inapproximability Results

In this section, we show that (1) problem LCS(unlimited, plain) cannot be approximated within ratio n^ϵ for any $\epsilon \in (0, \frac{1}{4})$, where n denotes the length of the longer input sequence and (2) problem LCS(crossing, plain) is MAX SNP-hard. To do that, we need the following well known problems.

Maximum Independent Set-B (MaxIS-B): Given a graph G in which every vertex has a degree at most B, find the largest independent set of G, *i.e.*, a subset of vertices in which no two vertices are connected in graph G.

Maximum Independent Set-Cubic (MaxIS-Cubic): Given a cubic graph G (*i.e.*, every vertex has a degree 3), find a largest independent set of G.

Lemma 1. [11, 3] MaxIS-B *is MAX SNP-complete when* $B \geq 3$.

The following lemma is necessary.

Lemma 2. MaxIS-Cubic *is MAX SNP-complete.*

Proof. The proof will be done by constructing an L-reduction [11] from MaxIS-3 to MaxIS-Cubic, using a technique from [5]. Given an instance of MaxIS-3, which is a graph $G(V, E)$ with $|V| = n$, we may assume that G is connected. Suppose that there are i vertices of degree 1, j vertices of degree 2 and $n - i - j$ vertices of degree 3. Let $opt(G)$ denote the size of a maximum independent set of G, then it follows trivially that $opt(G) \geq n/4$. Therefore, we have

$$i + j \leq 4 \cdot opt(G).$$

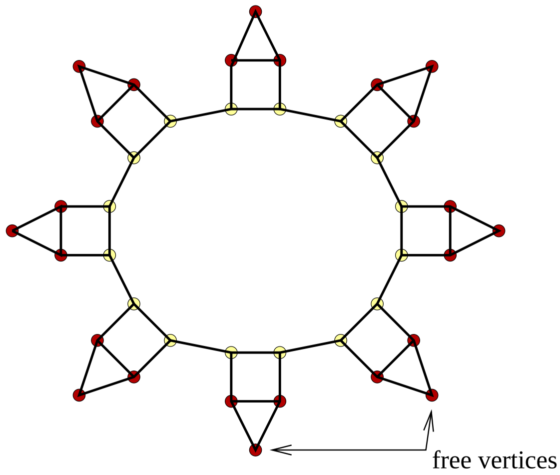

Fig. 3. The graph H.

We construct an instance of MaxIS-Cubic, a graph denoted by G', as follows. Construct $2i + j$ triangles of which each has two vertices connecting to two adjacent vertices in a cycle of size $2(2i + j)$ as shown in Figure 3. Denote this graph as H. Call the vertex of each triangle that is not adjacent to the cycle a *free* vertex in H. Clearly, the cycle itself has a maximum independent set of size $2i + j$. Since for each triangle the maximum independent set is a singleton, graph H has a maximum independent set of size exactly $2(2i + j)$. Moreover, we see that there is a maximum independent set of size $2(2i + j)$ for graph H consisting of no free vertices. The graph G' is then constructed from graphs G and H by connecting each degree 1 vertex in G to two unique free vertices in H and connecting each degree 2 vertex in G to one unique free vertex in H. Notice that resulting graph G' is indeed a cubic graph.

Suppose that V' is a maximum independent set for G (of size $opt(G)$), we may add a maximum independent set for H consisting of no free vertices into it to form an independent set for graph G'. This independent set would have a size of $k' = opt(G) + 2(2i + j)$. It follows that

$$opt(G') \geq opt(G) + 2(2i + j). \tag{1}$$

On the other hand, let V'' be an independent set for G' with $|V''| = k'$. Then deleting from V'' the vertices which are in graph H (the number of such vertices is at most $2(2i + j)$) will form an independent set for G of size k satisfying

$$k \geq k' - 2(2i + j). \tag{2}$$

Therefore, (1) becomes

$$opt(G') = opt(G) + 2(2i + j) \leq opt(G) + 4(i + j) \leq 17 \cdot opt(G), \tag{3}$$

and (2) becomes $k - opt(G) \geq k' - opt(G')$, or equivalently,

$$|k - opt(G)| \leq |k' - opt(G')|. \tag{4}$$

This completes the L-reduction.

Since problem MaxIS-Cubic is a special case of MaxIS-3, it is in the class MAX SNP. Hence, it is MAX SNP-complete. □

Theorem 2. *Problem* LCS(unlimited, plain) *cannot be approximated within ratio n^ϵ for any $\epsilon \in (0, \frac{1}{4})$, where n is the length of the longer input sequence.*

Proof. We will show that MaxIS, the unrestricted *Maximum Independent Set* problem, can be L-reduced to problem LCS(unlimited, plain). Since MaxIS cannot be approximated within ratio n^ϵ for any $\epsilon \in (0, \frac{1}{4})$ [2], where n is the number of vertices in the graph, the theorem follows.

Let graph $G(V, E)$, where $V = \{1, \ldots, n\}$, be an instance of MaxIS. Without loss of generality, we assume that the graph G is connected. The instance I of LCS(unlimited, plain) consists of two sequences defined as follows: $S_1 = \mathsf{a}^n$ with $P_1 = E$ and $S_2 = \mathsf{a}^n$ with $P_2 = \emptyset$. It is clear then that an independent set $V' = \{v_{i_1}, \ldots, v_{i_k}\}$ one-to-one corresponds to a arc-preserving common subsequence consisting of the i_1th, …, i_kth a's from sequence S_1. That is, LCS(unlimited, plain) in fact includes MaxIS as a subproblem. □

As a corollary, we have

Corollary 2. *Problem* LCS(unlimited, chain)*, problem* LCS(unlimited, nested)*, and problem* LCS(unlimited, unlimited) *cannot be approximated within ratio n^ϵ for any $\epsilon \in (0, \frac{1}{4})$, where n is the length of the longer input sequence.*

Theorem 3. *Problem* LCS(crossing, plain) *is MAX SNP-hard.*

Proof. To show its MAX SNP-hardness, we will L-reduce MaxIS-Cubic to problem LCS(crossing, plain). Given a cubic graph $G(V, E)$, let $n = |V|$. For each vertex $u \in V$, construct a segment T_u of letters aaaabbccc. Sequence S_1 is obtained by concatenating the n segments $T_u, u \in V$. The arc set P_1 on sequence

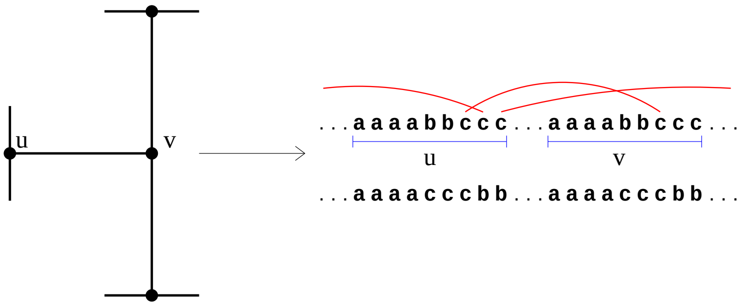

Fig. 4. Instance I constructed from cubic graph G.

S_1 is constructed as follows: Whenever there is an edge $(u, v) \in E$, introduce an arc between a letter c from T_u to a letter c from T_v, ensuring that each letter c is used only once. Sequence S_2 is obtained by concatenating n identical segments of aaaacccbb and its arc set $P_2 = \emptyset$. This constitutes an instance I of problem LCS(crossing, plain). See Figure 4 for an illustration.

Suppose that graph G has a maximum independent set V' with cardinality k. We put four letters a and three letters c from each $T_u, u \in V'$ into sequence Y and put four letters a and two letters b from each $T_u, u \notin V'$ into sequence Y. Clearly, sequence Y is a common subsequence of both S_1 and S_2. From the independency of V', we know that sequence Y inherits no arcs from P_1, and thus it is an arc-preserving common subsequence. Let $opt(I)$ denote the length of an longest arc-preserving common subsequence of S_1 and S_2, and $opt(G)$ denote the cardinality of a maximum independent set of graph G. It follows from the above that

$$opt(I) \geq opt(G) + 6n. \tag{5}$$

On the other hand, suppose that we have an arc-preserving common subsequence Y of length k' for (S_1, P_1) and (S_2, P_2). We observe first that Y inherits no arcs from P_1 since $P_2 = \emptyset$. Define a match between letters from the two segments at the same position in S_1 and S_2, *i.e.*, $S_1[9l+1, 9l+9]$ and $S_2[9l+1, 9l+9]$ for some $1 \leq l \leq n$, to be *canonical*. It is easy to see that the abundance of letters a in both S_1 and S_2 allow us to consider only common subsequences Y that induce mappings M consisting of only canonical matches. *I.e.*, we assume that for each $(i, j) \in M$, $i, j \in [9l + 1, 9l + 9]$ for some l.

Consider the canonical matches defined by the mapping M (induced by Y) between segments $S_1[9l+1, 9l+9]$ and $S_2[9l+1, 9l+9]$ for every l. Clearly, (i) the four a's should be matched and (ii) if a b is matched then no c's can be matched and vice versa. If only one or two c's are matched, we modify M (and thus Y) as follows: unmatch the c's and match the two b's. Clearly, this modification doesn't make the solution worse. We repeat this modification until the canonical matches between each segment pair contains either all three c-matches or two b-matches. Denote the final mapping as M'.

We now define a subset V' of vertices of G as follows: for every segment T_u in sequence S_1, if all its three c's are matched M', we put u in V'. By the construction of arc set P_1, no pair of vertices in V' are connected in graph G and hence V' is an independent set for G. Let $k = |V'|$, then we have $k \geq k' - 6n$. Since G is a cubic graph, $n/4 \leq opt(G) \leq n/2$ and inequality (5) becomes

$$opt(I) = opt(G) + 6n \leq 25 \cdot opt(G). \tag{6}$$

Hence $k \geq k' - 6n = k' - (opt(I) - opt(G))$, which is equivalent to

$$|k - opt(G)| \leq |k' - opt(I)|. \tag{7}$$

Inequalities (6) and (7) show that our reduction is an L-reduction, and thus problem LCS(crossing, plain) is MAX SNP-hard. □

As a corollary, we have

Corollary 3. *Problem* LCS(crossing, chain), *problem* LCS(crossing, nested), *and problem* LCS(crossing, crossing) *are all MAX SNP-hard.*

We note in passing that there is a similar, but more restrictive, LCS definition [14], where in addition to our condition that, for any (i_1, j_1) and (i_2, j_2) in the mapping, $(i_1, i_2) \in P_1$ if and only if $(j_1, j_2) \in P_2$, it also requires that for any (i_1, j_1) in the mapping, if $(i_1, i_2) \in P_1$ then, for some j_2, (i_2, j_2) is in the mapping, and if $(j_1, j_2) \in P_2$ then, for some i_2, (i_2, j_2) is in the mapping. For that definition LCS(crossing, crossing) is NP-hard and LCS(crossing, nested) is solvable in polynomial time [14].

4 Dynamic Programming Algorithm for LCS(nested, plain)

Given a pair (S_1, P_1) and $(S_2, \emptyset)$ of arc-annotated sequences with P_1 being nested, denote $n = |S_1|$ and $m = |S_2|$. For any arc $u \in P_1$, let u_l and u_r denote its *left* and *right* endpoints, respectively. We also use $u(i)$ to denote the arc in P_1 incident on position i of sequence S_1. If $u(i)$ does not exist, then we call position i *free*. We formulate a recurrence relation for computing the length of a longest arc-preserving common subsequence for the pair (S_1, P_1) and $(S_2, \emptyset)$.

Let $DP(i, i'; j, j')$, where $1 \leq i \leq i' \leq n$ and $1 \leq j \leq j' \leq m$, denote the length of a longest arc-preserving common subsequence for the pair $(S_1[i, i'], P_1[i, i'])$ and $(S_2[j, j'], \emptyset)$. Define $\chi(S_1[i], S_2[j]) = 1$ if $S_1[i] = S_2[j]$, or 0 otherwise. Our algorithm is a two-step dynamic programming.

The first step of our algorithm is as follows:

For each arc $(i_1, j_1) \in P_1$, we perform the following computation.
Phase 1: If $j_1 - i_1 > 1$, then let $i = i_1 + 1$ and i' be any position in $[i, j_1 - 1]$, such that for any arc $(k, k') \in P_1$, either $[i, i'] \subseteq [k, k']$, or $[k, k'] \subseteq [i, i']$, or $[i, i'] \cap [k, k'] = \emptyset$.

If i' is free, then

$$DP(i,i';j,j') = \max \begin{cases} DP(i,i'-1;j,j'-1) + \chi(S_1[i'],S_2[j']), \\ DP(i,i'-1;j,j'), \\ DP(i,i';j,j'-1). \end{cases}$$

If i' is not free (i.e., $i' = u(i')_r$) and $i \neq u(i')_l$, then

$$DP(i,i';j,j') = \max_{j \le j'' \le j'} \Big\{ DP(i,u(i')_l - 1; j, j''-1) + DP(u(i')_l, i'; j'', j') \Big\}.$$

Phase 2:

$$DP(i_1,j_1;j,j') = \max \begin{cases} DP(i_1+1,j_1-1;j+1,j') + \chi(S_1[i_1],S_2[j]), \\ DP(i_1+1,j_1-1;j,j'-1) + \chi(S_1[j_1],S_2[j']), \\ DP(i_1+1,j_1-1;j,j'), \\ DP(i_1,j_1;j,j'-1), \\ DP(i_1,j_1;j+1,j'). \end{cases}$$

Note that for any two arcs (i_1,j_1) and (i_2,j_2) in P_1, let s_1 and s_2 be the sets of i' satisfying conditions of *Phase 1*, then s_1 and s_2 are disjoint. Therefore the total number of entries for $DP(i,i';j,j')$ is $O(nm^2)$. This means that the first step can be done in (nm^3).

The second step of the algorithm is very similar to *Phase 1* of the first step.

Let i' be any position in $[1,n]$, such that for any arc $(k,k') \in P_1$, either $[1,i'] \subseteq [k,k']$, or $[k,k'] \subseteq [i,i']$, or $[1,i'] \cap [k,k'] = \emptyset$.

If i' is free, then

$$DP(1,i';j,j') = \max \begin{cases} DP(1,i'-1;j,j'-1) + \chi(S_1[i'],S_2[j']), \\ DP(1,i'-1;j,j'), \\ DP(1,i';j,j'-1). \end{cases}$$

If i' is not free (i.e., $i' = u(i')_r$) and $u(i')_l \neq 1$, then

$$DP(1,i';j,j') = \max_{j \le j'' \le j'} \Big\{ DP(1,u(i')_l - 1; j, j''-1) + DP(u(i')_l, i'; j'', j') \Big\}.$$

It is clear that this step can be done in $O(nm^3)$ too. Therefore the time complexity of the algorithm is $O(nm^3)$. By using a standard back-tracing technique, we can find a longest arc-preserving common subsequence for the pair (S_1,P_1) and $(S_2,\emptyset)$ from the table entries. Therefore, we have the following theorem, which answers an open question in [6,7].

Theorem 4. *Problem* LCS(nested, plain) *can be solved in time* $O(nm^3)$.

We may extend the above construction to solve problem LCS(nested, chain) as well as to problem LCS(crossing, nested) in which the cutwidth of the first sequence is upper bounded by a constant k. For example, for problem LCS(nested, chain), we expand the entry $DP(i,i';j,j')$ to entry $DP(i,i';j,j';\alpha,\beta)$, where α

and β denote the positions between the jth and j'th on the second sequence $S_2[j, j']$ that the letters therein cannot be matched to any letter in the first sequence $S_1[i, i']$. More specifically, for a given pair of j and j', there might be an arc in P_2 crossing (not connecting to) position j (j', respectively) and its right (left, respectively) endpoint lying inside $[j+1, j'-1]$. Since we want to compute the arc-preserving common subsequence of $S_1[i, i']$ and $S_2[j, j']$ independently, we have to register if this endpoint can be included into the common subsequence. If it cannot, we will then let α (β, respectively) denote this right (left, respectively) endpoint of the arc crossing position j (j', respectively). In the other cases, α (β, respectively) denotes nothing (represented by $-$). The recurrence relation needs modifications to fit the computation. For example, in *Phase 1* and i' is free, let ρ denote the rightmost position in $S_2[j, j'-1]$ that is not α, neither β. If $j' = u(j')_l$ then we need to compute

$$DP(i, i'; j, j'; \alpha, -) = \max \begin{cases} DP(i, i'-1; j, \rho; \alpha', -) + \chi(S_1[i'], S_2[j']), \\ DP(i, i'-1; j, j'; \alpha, -), \\ DP(i, i'; j, \rho; \alpha', -), \end{cases}$$

and

$$DP(i, i'; j, j'; \alpha, j') = DP(i, i'; j, \rho; \alpha', -),$$

where $\alpha' = \alpha$ if $\alpha < \rho$, $-$ otherwise;

If $j' = u(j')_r (\neq \alpha)$ then we need to compute

$$DP(i, i'; j, j'; \alpha, -) = \max \begin{cases} DP(i, i'-1; j, \rho; \alpha, \beta') + 1, \text{ if } S_1[i'] = S_2[j'], \\ DP(i, i'-1; j, j'; \alpha, -), \\ DP(i, i'; j, j'-1; \alpha, -), \end{cases}$$

where $\beta' = u(j')_l$ if $j \leq u(j')_l < \rho$, $-$ otherwise; and

$$DP(i, i'; j, j'; \alpha, j') = DP(i, i'; j, j'-1; \alpha, -);$$

In the other cases, we need to compute

$$DP(i, i'; j, j'; \alpha, \beta) = \max \begin{cases} DP(i, i'-1; j, \rho; \alpha', \beta') + \chi(S_1[i'], S_2[j']), \\ DP(i, i'-1; j, j'; \alpha, \beta), \\ DP(i, i'; j, \rho; \alpha', \beta'), \end{cases}$$

where $\alpha' = \alpha$ if $\alpha < \rho$, $-$ otherwise; $\beta' = \beta$ if $\beta < \rho$, $-$ otherwise.

The case in which i' is not free, the *Phase 2* and the second step of the algorithm can be similarly modified. Notice that each entry $DP(i, i'; j, j')$ is expanded to at most four entries. Again, each entry can be computed in $O(m)$ time from its "preceding" entries. It follows that problem LCS(nested, chain) can be solved in time $O(nm^3)$. For problem LCS(crossing, nested), each entry $DP(i, i'; j, j')$ is expanded by introducing k pairs of parameters, α_1, β_1, α_2, β_2, $\cdots$, α_k, β_k, to denote the $2k$ possible positions in sequence $S_1[i, i']$ that the letters therein cannot be matched to any letter in sequence $S_2[j, j']$. These expanded entries can be used to solve problem LCS(crossing, nested) where the cutwidth of the first input sequence is upper bounded by a constant k.

Theorem 5. *Problem* LCS(nested, chain) *can be solved in time* $O(nm^3)$. *The restricted version of problem* LCS(crossing, nested) *where the cutwidth of the first input sequence is upper bounded by a constant* k *can be solved in time* $O(4^k nm^3)$.

We note in passing that the restricted version of problem LCS(crossing, crossing) where both input sequences have cutwidth upper bounded by k is solvable in time $O(9^k nm)$ [6].

5 Concluding Remarks

In this paper, we have considered the longest common subsequence problem for arc-annotated sequences with arc structures of various types, and presented some new algorithmic and complexity results. We leave as an open problem to determine the computational complexity of problem LCS(nested, nested).

Acknowledgment. We thank Professor Paul Kearney for many helpful discussions.

References

1. V. Bafna, S. Muthukrishnan and R. Ravi, Computing similarity between RNA strings, *DIMACS Technical Report* 96-30, 1996.
2. M. Bellare, O. Goldreich and M. Sudan, Free bits, PCPs and non-approximability - towards tight results, *SIAM Journal on Computing*, 27(1998), 804-915.
3. P. Berman and T. Fujito, On approximation properties of the independent set problem for degree 3 graphs, in *Proceedings of 4th International Workshop on Algorithms and Data Structures (WADS'95)*, LNCS 955, pp. 449–460.
4. F. Corpet and B. Minchor, RNAling program: alignment of RNA sequences using both primary and secondary structures, *Computer Applications in the Bio-sciences*, 10(1994), 389–399.
5. D.Z. Du, B. Gao and W. Wu, A special case for subset interconnection designs, *Discrete Applied Mathematics*, 78(1997), 51–60.
6. P.A. Evans, *Algorithms and Complexity for Annotated Sequence Analysis*, Ph. D Thesis, University of Victoria, 1999.
7. P.A. Evans, Finding common subsequences with arcs and pseudoknots, in *Proceedings of 10th Annual Symposium on Combinatorial Pattern Matching (CPM'99)*, LNCS 1645, pp. 270–280.
8. D. Goldman, S. Istrail and C.H. Papadimitriou, Algorithmic aspects of protein structure similarity, *Proc. IEEE 40th Annual Conference of Foundations of Computer Science (FOCS'99)*, 1999.
9. D.S. Hirschberg, The longest common subsequence problem. *Ph.D. Thesis*, Princeton University, 1975.
10. H. Lenhof, K. Reinert and M. Vingron, A polyhedral approach to RNA sequence structure alignment, in *Proceedings of the Second Annual International Conference on Computational Molecular Biology (RECOMB'98)*, 153–159.
11. C.H. Papadimitriou and M. Yannakakis, Optimization, approximation, and complexity classes, *Journal of Computer and System Science*, 43(1991), 425–440.

12. D. Sankoff, Simultaneous solution of the RNA folding, alignment, and protosequence problems, *SIAM Journal on Applied Mathematics*, 45(1985), 810–825.
13. R.A. Wagner and M.J. Fischer, The string-to-string correction problem. Journal of the ACM, 21(1)(1974), 168-173.
14. K. Zhang, L. Wang and B. Ma, Computing similarity between RNA structures, in *Proceedings of 10th Annual Symposium on Combinatorial Pattern Matching (CPM'99)*, LNCS 1645, pp. 281–293.