RNA multiple structural alignment with longest common subsequences ⋆

Sergey Bereg [a], Marcin Kubica [b], Tomasz Waleń [b], Binhai Zhu [c,*],

[a] *Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083-0688, USA.*

[b] *Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warszawa, Poland.*

[c] *Department of Computer Science, Montana State University, Bozeman, MT 59717-3880, USA.*

Abstract

In this paper, we present a new model for RNA multiple sequence structural alignment based on the *longest common subsequence*. We consider both the off-line and on-line cases. For the off-line case, i.e., when the longest common subsequence is given as a linear graph with n vertices, we first present a polynomial $O(n^2)$ time algorithm to compute its maximum nested loop. We then consider a slightly different problem—the Maximum Loop Chain problem and present an algorithm which runs in $O(n^5)$ time. For the on-line case, i.e., given m RNA sequences of lengths n, compute the longest common subsequence of them such that this subsequence either induces a maximum nested loop or the maximum number of matches, we present efficient algorithms using dynamic programming when m is small.

Key words: RNA multiple structure alignment, longest common subsequence, dynamic programming

⋆ This research is partially supported by EPSCOR Visiting Scholar's Program and MSU Short-term Professional Development Program.

* Corresponding author: bhz@cs.montana.edu

Email: besp@utdallas.edu(Sergey Bereg), kubica@mimuw.edu.pl(Marcin Kubica), walen@mimuw.edu.pl(Tomasz Waleń), bhz@cs.montana.edu(Binhai Zhu)

1 Introduction

In the study of noncoding RNA (*ncRNA*), it is well known that the corresponding nucleotides are very active among genomic DNA. There are four such (polymers of) nucleotides: A, C, G and U. Different from regular genes, ncRNAs are not translated into protein and they fold directly into secondary and tertiary structures and the stability of the foldings are mainly determined by A-U, C-G and G-U bonds [7].

However, it is still not completely known how such a ncRNA folds into secondary and tertiary structures. One of the methods is to take a multiple sequence of ncRNAs and investigate their common folding patterns or secondary structures [22,23,5]. In [5], it is proposed that the largest common nested linear subgraph of m given linear graphs (induced by m ncRNA sequences of length n, such that the edges are those non-adjacent A-U, C-G, G-U bonds) presents a solution for this problem. This problem is NP-complete [5], and there is a factor-$O(\log n)$ approximation for this problem [13].

In this paper, we follow the general methodology of [5]. However, we think that computing largest common nested linear subgraph cannot perfectly solve the problem in many situations. For example, if we have two ncRNA sequences: AGUU and CAGG, even though they induce the same largest common nested linear subgraph, the corresponding bonds and letters are completely different. (A letter cannot form a *bond* or *match* with a neighboring letter.)

The above idea forms the basis of our research. In this paper, we propose to use the Longest Common Subsequence (LCS) of m given ncRNA sequences as the basis to tackle this problem. We consider two general cases: off-line and on-line cases. In the off-line case, the LCS is already given and we want to find meaningful properties of such a LCS, namely, whether this LCS admits a special kind of fold. In the on-line case, we want to compute the LCS which admits certain kind of folding.

In general, the longest common subsequence of two sequences is not unique. Rick [18] developed an algorithm for finding *all* longest common subsequences. The number of longest common subsequences can be quite large. Greenberg [9] proved an exponential lower bound for the maximum number of distinct longest common subsequences of two sequences of length n. Therefore, finding longest common subsequences suitable for various biological applications is an important problem. The goal of this paper is to find a longest common subsequence satisfying useful properties—to maximize the A-U, C-G and G-U bonds in several different ways. A related work is on identifying a (sub)string which is close to a set of given strings [14,15] and close to a set of 'bad' strings and far from a set of 'good' strings [6]. A constrained longest common

subsequence problem was studied recently by Tsai [21] and Chin et al.[1]. In this problem three strings X, Y and P are given, one wants to find the longest subsequence of X and Y such that P is its subsequence.

In this paper, we mainly focus on three kinds of folding: maximum nested loop, maximum loop chains and maximum number of total matches. For the off-line case the problem is more of a graph theoretical one and we present polynomial time solutions. For the on-line case, the problem is NP-complete in general as computing LCS of multiple sequences, even without any other constraint, is NP-complete [16]. We try to present efficient algorithms for cases when m is relatively small.

2 Preliminaries

In this section we first present necessary definitions.

Throughout this paper, in a given sequence over $\{A, C, G, U\}$, two non-adjacent characters (letters) $a, b \in \{A, C, G, U\}$ *match* or form a *bond* if $\{a, b\} = \{A, U\}$, or $\{a, b\} = \{C, G\}$, or $\{a, b\} = \{G, U\}$. Given a sequence $t = a_1 a_2 ... a_n$, $a_i \in \{A, C, G, U\}$, the corresponding linear graph $G(t)$ is defined as follows. The vertices of $G(t)$ are integers $1, 2, ..., n$ and there is an edge between i and j $(j > i + 1)$ if a_i and a_j match each other. For the obvious reason a_i cannot match a_{i+1} for $i = 1, 2, ..., n - 1$. In other words, there is no edge between i and $i + 1$ for $i = 1, 2, ..., n - 1$. This linear graph certainly characterizes the general folding possibilities of all the letters in t. In [8], a similar graph called *contact map graph* is also used for identifying protein structure similarity.

Given two edges e_1, e_2 in $G(t)$ and the intervals $I_1 = [a, b], I_2 = [c, d]$ spanned by them, we say e_1 intersects e_2 if exactly one of a, b lies on $[c, d]$ and vice versa. Therefore, if e_1 does not intersect e_2, then either I_1 and I_2 are disjoint or I_1 is contained in I_2 (assuming I_2 is longer). Moreover, if $c < a$ and $b < d$, then we say that I_2 *surrounds* I_1. A set of edges $e_1, e_2, ..., e_p$ in $G(t)$ form a *nested loop with depth* (or just *loop*) p if the intervals $I_1, I_2, ..., I_p$ spanned by $e_1, e_2, ...e_p$ surround one another, i.e., I_{j+1} surrounds I_j, for $j = 1, \ldots, p - 1$ (Figure 1(1)).

Given a linear graph $G(t)$, two loops *overlap* if all the edges in one loop L_1 intersect all the edges in the other loop L_2. Such an overlap is *legal* if no two edges from L_1, L_2 share the same vertex in $G(t)$. We say that L_1 and L_2 are *disjoint* if the edges from L_1 and L_2 are pairwise disjoint. We define a *chain of loops* (or *loop chains*) as a set of loops $L_1, L_2, ..., L_w$ such that L_i legally overlaps with L_{i+1}, and moreover L_i and L_{i+x} are disjoint, for $i \leq w-1, x > 1$. The motivation behind this is that a chain of (relatively deep) loops provide

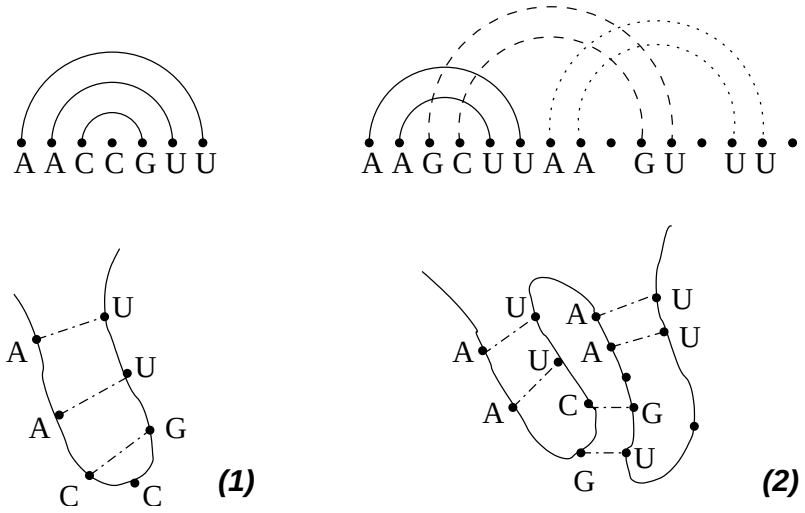

Fig. 1. An illustration of ncRNA folding with maximum loop and maximum loop chains.

a special kind of stable folding, also known as *pseudoknot* [19], for a given ncRNA sequence (Figure 1(2)).

In this paper, we propose to study several problems based on the Longest Common Subsequence (LCS). The LCS problem has been thoroughly studied in Hirschberg's PhD thesis [10]. Its application in computational biology dated back to 1960s [3,4]. Some other applications of LCS in computational biology can be found in [20,11]. Basically, for a set of m (m being a constant) sequences of length n, the corresponding LCS can be computed in $O(n^m)$ time. If m is not a constant, then the problem is NP-complete; moreover, if the alphabet is unbounded then it is difficult to find an approximation solution (in fact, it is as hard as approximating the Maximum Clique problem) [12].

3 The off-line case: when the LCS is already given

For the ncRNA multiple structural alignment problem, in general we want to compute a LCS with some additional constraints. In this section, we consider the off-line case when a LCS of some ncRNA sequences is already computed. The first problem is based on the idea that the (maximum) deepest nested loop is likely to occur in ncRNA folding (Fig. 1(1)). The second problem is based on the idea that a chain of loops is likely to fold compactly within some specified regions (Fig. 1(2)).

3.1 The Maximum Nested Loop problem

Given a sequence (which is the LCS of some ncRNA sequences) $t = a_1a_2...a_n, a_i \in \{A, C, G, U\}$, and the corresponding linear graph $G(t)$, compute the maximum or the deepest nested loop (MNL) in $G(t)$. We have the following theorem.

Theorem 1 *Given a sequence $t = a_1a_2...a_n, a_i \in \{A, C, G, U\}$, and the corresponding linear graph $G(t)$, the maximum nested loop can be computed in $O(n^2)$ time.*

PROOF. For $1 \leq i \leq j \leq n$, let $t_{i,j}$ denote the sequence $a_ia_{i+1} \dots a_j$. Let S be a two-dimensional array where $S[i, j]$ is the maximum depth of a nested loop in the sequence $t_{i,j}$. The values of the array S can be computed as follows. For any $1 \leq i \leq n$, $S[i, i] = 0$.

Suppose that, for $1 \leq i < j \leq n$, a_i and a_j match (which implies that $j > i+1$). Then there is a maximum nested loop of $t_{i,j}$ that contains the edge (a_i, a_j). Thus $S[i, j] = S[i + 1, j - 1] + 1$.

Suppose that a_i and a_j do not match. Then either a_i or a_j is not an endpoint of the outermost edge of a maximum nested loop of $t_{i,j}$. Thus, $S[i, j] = \max(S[i+1, j], S[i, j-1])$. We summarize all the cases in pseudo-code (Algorithm MNL).

The depth of maximum nested loop in the input sequence is $S[1, n]$. In order to compute the nested loop we store auxiliary arrays A and B such that $(A[i, j], B[i, j])$ is the outermost edge of a maximum nested loop of $t_{i,j}$. The values $A[i, j]$ and $B[i, j]$ can be updated at the time when $S[i, j]$ is updated.

The algorithm clearly takes $O(n^2)$ in the worst case. □

A slightly different $O(n^3)$ time result on loop matching in programming language research was known long time ago [17]. That result has been used in RNA folding [5]. Although the Maximum Nested Loop problem is slightly more easier to solve, in biology it could be a very important subroutine. In ncRNAs, A-U, C-G and G-U bonds almost always occur in a nested fashion [5]; so finding such maximum nested loop is very meaningful, at least it will allow biologists to try different alternatives in folding.

```
Algorithm MNL
for l = 1 to n
  for i = 1 to n − l + 1
    j := i + l − 1
    if l = 1 then
      S[l, l] := 0
    elseif a_i and a_j match then
      if j − i > 1 then S[i, j] := S[i + 1, j − 1] + 1
      else S[i, j] := 1
      A[i, j] := i
      B[i, j] := j
    elseif S[i + 1, j] > S[i, j − 1] then
      S[i, j] := S[i + 1, j]
      A[i, j] := A[i + 1, j]
      B[i, j] := B[i + 1, j]
    else
      S[i, j] := S[i, j − 1]
      A[i, j] := A[i, j − 1]
      B[i, j] := B[i, j − 1]
```

3.2 The Maximum Loop Chain problem

In this subsection we investigate a slightly different problem. When a ncRNA sequence (with its corresponding linear graph) and a set of nested loops are given, we have the following problem of computing the *maximum loop chain*.

The **Maximum Loop Chain** (MLC) problem: Given a ncRNA sequence $t = a_1 a_2 \dots a_n$, $a_i \in \{A, C, G, U\}$, the corresponding linear graph $G(t)$ and a set N of nested loops in $G(t)$, compute a loop chain out of these loops such that its size is maximized. The *size* of a loop chain is the sum of depths of its loops.

We assume that the input nested loops are given together with its depth. As we will discuss a bit later, there could be at most $O(n^4)$ number of nested loops following our setting, so N is at most $O(n^4)$. Alternatively, if the loops are not given in advance then we can generate the set of all possible nested loops in $O(n^4)$ time. We have the following theorem.

Theorem 2 *The Maximum Loop Chain problem can be solved in $O(n^5)$ time.*

PROOF. The algorithm is divided into two phases. In the first phase we calculate an array P describing maximum depths of nested loops binding given

parts of the ncRNA sequence. We define $P[i, j, k, l]$ as the maximum depth of a nested loop of edges e such that the left endpoint of e is in $[i, j]$ and the right endpoint of e is in $[k, l]$, see Fig. 2 for an example. The values of the 4-dimensional array $P[..]$ can be computed using dynamic programming as follows.

$$P[i,j,k,l] = \begin{cases} 0 & \text{if } i > j \text{ or } k > l \\ 1 + P[i, j-1, k+1, l] & \text{if } a_j \text{ matches } a_k \\ \max(P[i, j-1, k, l], P[i, j, k+1, l]) & \text{otherwise.} \end{cases}$$

The running time for computing $P[..]$ is $O(n^4)$ time.

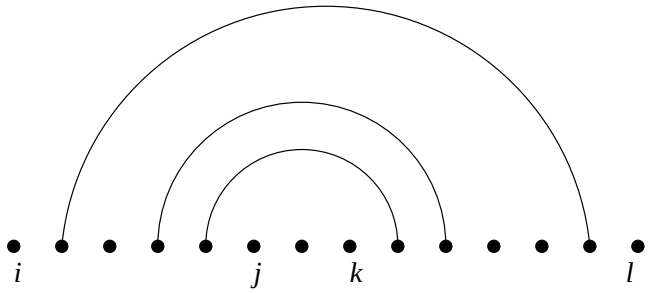

Fig. 2. A loop of $P[i, j, k, l]$.

In the second phase we compute an array C containing the maximum sizes of loop chains of different shapes. We define $C[k, l, m]$ as the maximum size of a loop chain $C_1, C_2, \ldots, C_t$ such that (i) the last loop contains at least one edge with $|C_t| \geq 1$ if $C[k, l, m] > 0$, and (ii) for every edge (a, b) of the last loop C_t, $a \leq k$ and $l \leq b \leq m$, and (iii) for every edge (a, b) of loops $C_1, C_2, \ldots, C_{t-1}$, $b < k$. Refer to Fig. 3 for an example.

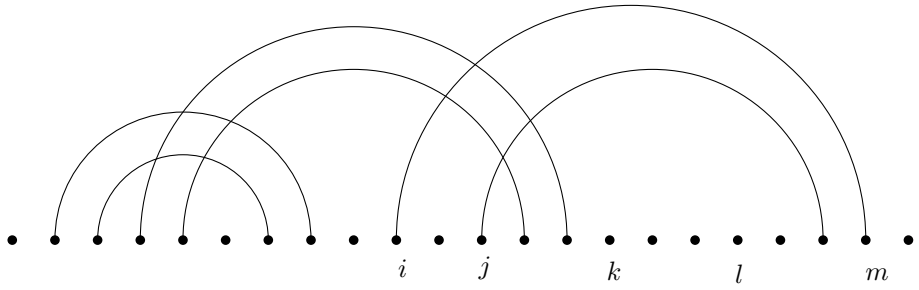

Fig. 3. A loop chain of $C[k, l, m]$.

Let k, l, m be integers such that $1 \leq k \leq l \leq m \leq n$. Let $L_1, L_2, \ldots, L_t$ be a loop chain of size $C[k, l, m]$ satisfying the conditions (i), (ii) and (iii) above. Suppose that the edges of the loop L_t have left endpoints in $[i + 1, j - 1]$ and right endpoints in $[l, m]$. Then the number of edges in L_t is $P[i+1, j-1, l, m]$. The total size of loops $L_1, L_2, \ldots, L_{t-1}$ is $C[i, j, k]$. Therefore $C[k, l, m]$ can be computed in $O(n^2)$ time for the triple (k, l, m). The following pseudo-code shows the computation of all values of the array $C[..]$.

```
for k = 1 to n − 1 do
  for l = k + 1 to n do
    for m = l to n do begin
      C[k, l, m] := P[1, k, l, m];
      for i = 1 to k − 1 do
        for j = i + 1 to k do
          C[k, l, m] := max(C[k, l, m], C[i, j, k] + P[i + 1, j − 1, l, m])
```

The maximum integer stored in array C is the size of MLC. The actual nested loop chain can be reconstructed from arrays C and P in $O(n^3)$ time. So, MLC problem can be solved in $O(n^5)$ total time complexity. □

Please note, that in the above algorithm we do not exploit the fact that the linear graph is generated by a ncRNA sequence. Therefore, this algorithm works for arbitrary linear graphs.

Presented algorithm works for a given set of nested loops. However, it can be easily adapted to solve MLC problem for the set of all possible nested loops. The following procedure calculates array P for all nested loops.

```
for j = 1 to n − 1 do
  for k = j + 1 to n do
    for i = j downto 1 do
      for l = k to n do
        if there is an edge (i, l) then
          P[i, j, k, l] := P[i + 1, j, k, l − 1] + 1
        else
          P[i, j, k, l] := max(P[i + 1, j, k, l], P[i, j, k, l − 1]);
```

Clearly, it is running in $O(n^4)$ time, so the overall time complexity of the algorithm remains $O(n^5)$.

4 The on-line case: when the LCS is not given

In this section, we study the problem when the LCS of a set of m ncRNA sequences is not given in advance. We study two versions of this general problem: LCSMNL and LCSBM. As computing LCS for multiple sequences is in general NP-complete, both of these problems are NP-complete. We are interested in efficient solutions when m is relatively small. Recall that there might be too many LCS's for some given sequences, our goal is to identify a LCS with some useful property.

4.1 LCSMNL

We first consider the problem of computing the *longest common subsequence with maximum nested loop* (LCSMNL).

LCSMNL problem. Given a set S of strings $s_1, s_2, \ldots, s_m$, each of length n, where $s_i = a_{i1}a_{i2}\ldots a_{in}$ and $a_{ij} \in \{A, C, G, U\}$, compute the longest common subsequence $s = g_1 g_2 \ldots g_K$ of $s_1, s_2, ..., s_m$ such that the maximum nested loop induced by (i.e., reconstructed from) s is maximized.

Theorem 3 *The* **LCSMNL** *problem can be solved in* $O(n^{2m})$ *time. When* $m = 2$*, the problem can be solved in* $O(n^4)$ *time.*

PROOF. We show the algorithm for $m = 2$ only. It is straightforward to extend it to general $m \geq 2$. We use a simplified notation $s_1 = a_1 a_2 \ldots a_n$ and $s_2 = b_1 b_2 \ldots b_n$. Let $a_{i,j}$, resp. $b_{i,j}$, denote the substring $a_i a_{i+1} \ldots a_j$, resp. $b_i b_{i+1} \ldots b_j$. We store four four-dimensional arrays L, D, A, B defined as follows. For $1 \leq i \leq j \leq n$ and $1 \leq k \leq l \leq n$, let $\Xi(i, j, k, l)$ denote the set of all longest common subsequences of $a_{i,j}$ and $b_{k,l}$. Then

- $L[i, j, k, l]$ is the length of the longest common subsequence of $a_{i,j}$ and $b_{k,l}$,
- $D[i, j, k, l]$ is the maximum depth of the maximum nested loop induced by a sequence $\xi \in \Xi(i, j, k, l)$,
- $(A[i, j, k, l], B[i, j, k, l])$ is an outermost edge of a maximum nested loop induced by a sequence $\xi \in \Xi(i, j, k, l)$.

A, B are used to retrieve the actual loop edges and the update of A, B is obvious from the following algorithm and is omitted. The items of the array $L[..]$ can be computed in the same way as the computation of longest common subsequences. The value of $D[i, j, k, l]$ can be computed as in the pseudo-code shown in Algorithm LCSMNL. Notice that we need to use the values of $L[..]$ in the algorithm to guarantee that only the nested loops among some longest common subsequence can be computed. The theorem follows. □

The optimal solution (depth of the maximum nested loop of a LCS of s_1, s_2) is found in $D[1, n, 1, n]$. It is easy to see that we can in fact allow s_1 and s_2 to have different lengths.

Algorithm LCSMNL
Compute $L[-,-,-,-]$
// Compute $D[..]$
for $l_1 = 1$ **to** n
  for $i = 1$ **to** $n - l_1 + 1$
    $j := i + l_1 - 1$
    for $l_2 = 1$ **to** n
      for $k = 1$ **to** n
        $l := k + l_2 - 1$; $d := 0$
        if $L[i,j,k,l] = L[i,j-1,k,l]$ **then**
          $d := \max(d,\ D[i,j-1,k,l])$;
        if $L[i,j,k,l] = L[i,j,k,l-1]$ **then**
          $d := \max(d,\ D[i,j,k,l-1])$;
        if $L[i,j,k,l] = L[i+1,j,k,l]$ **then**
          $d := \max(d,\ D[i+1,j,k,l])$;
        if $L[i,j,k,l] = L[i,j,k+1,l]$ **then**
          $d := \max(d,\ D[i,j,k+1,l])$;
        if ($a_j = b_l$ **and** $L[i,j,k,l] = L[i,j-1,k,l-1] + 1$) **then**
          $d := \max(d,\ D[i,j-1,k,l-1])$;
        if ($a_i = b_k$ **and** $L[i,j,k,l] = L[i+1,j,k+1,l] + 1$) **then**
          $d := \max(d,\ D[i+1,j,k+1,l])$;
        if ($a_i = b_k$ **and** $a_j = b_l$ **and** a_i matches a_j
            and $L[i,j,k,l] = L[i+1,j-1,k+1,l-1] + 2$) **then**
          $d := \max(d,\ D[i+1,j-1,k+1,l-1] + 1)$;
        $D[i,j,k,l] := d$;

4.2 LCSBM

Finally, we study the problem of computing a LCS which induces the maximum number of total matches, or *longest common subsequence with bounded matches* (LCSBM).

LCSBM problem. Given a set S of strings $s_1, s_2, \ldots, s_m$, each of length n, where $s_i = a_{i1}a_{i2}\ldots a_{in}$ and $a_{ij} \in \{A, C, G, U\}$, compute the longest common subsequence $s = g_1 g_2 \ldots g_K$ of $s_1, s_2, ..., s_m$ such that the total number of matches among non-adjacent g_i and g_j is maximized.

We assume that $m = 2$, and the algorithm can be easily extended to $m \geq 3$. We use a simplified notation $s_1 = a_1 a_2 \ldots a_n$ and $s_2 = b_1 b_2 \ldots b_n$. Let $b : \{A, C, G, U\} \rightarrow \{A, C, G, U\}$ be the mapping between a character and another one such that they form a bond. Then, $b(A) = U, b(C) = G, b(G) = U, b(U) = A$ or G, and vice versa. Adding another character $x \in \{A, C, G, U\}$

to the end of a string, x induces a number of matches to its non-adjacent characters following the above setting.

Let $LCS[i, j]$ be the length of the longest common subsequence of the sequences $a_1 a_2 \ldots a_i$ and $b_1 b_2 \ldots b_j$. The array $LCS[i, j]$ can be computed in $O(n^2)$ time [2].

Let a, c, g and u be integers and x be any letter from $\{A, C, G, U\}$. A sequence s is called (a, c, g, u, x)-*sequence* if s contains a letters A, c letters C, g letters G, u letters U and the last letter of s is x.

We use 6-dimensional array $M[n, n, n, n, n, 4]$ whose elements are defined as follows. Let $1 \leq i, j, k \leq n$ and $x \in \{A, C, G, U\}$ and $l = LCS[i, j]$. Let $0 \leq a, c, g \leq l$ be any integers such that $u = l - a - c - g \in [0, l]$. The value $M[i, j, a, c, g, x]$ is -1 if there is no (a, c, g, u, x)-sequence s of length l that is the common subsequence of $a_1 a_2 \ldots a_i$ and $b_1 b_2 \ldots b_j$ such that the last letter of s is x. If such a (a, c, g, u, x)-sequence s, which is a common subsequence of $a_1 a_2 \ldots a_i$ and $b_1 b_2 \ldots b_j$, exists, then $M[i, j, a, c, g, x]$ stores the maximum number of matches in s. The pseudo-code, which is a straightforward implementation of the above idea, is listed as Algorithm LCSBM.

Eventually, the optimal solution (i.e., the maximum number of matches of a LCS of s_1 and s_2) can be searched in all the cells $M[n, n, -, -, -, -]$. (In fact, by searching $M[n, n, -, -, -, -]$ we can also find many 'close to optimal' matches, which might be useful as well.) The actual LCS realizing the corresponding maximum matches can be reconstructed using standard methods.

Algorithm LCSBM
Initialize $M[i, j, i_1, j_1, k_1, x]$ to -1 if $i \geq 1$ or $j \geq 1$ and to 0 if $i = 0$ or $j = 0$.
Compute $LCS[-, -]$
for $i = 1$ **to** n
  for $j = 1$ **to** n
    $l := LCS[i, j]$
    for $a = 0$ **to** l // a is the number of A in LCS
      for $c = 0$ **to** $l - a$ // c is the number of C in LCS
        for $g = 0$ **to** $l - a - c$ // g is the number of U in LCS
          $u := l - a - c - g$ // u is the numbers of U in LCS
          if $a_i = b_j$ and a_i is counted at least one time in (a, c, g, u) **then**
            Let (a', c', g', u') be the the same numbers as (a, c, g, u)
                with one letter a_i removed
            for each $x \in \{A, C, G, U\}$
              if $M[i, j, a', c', g', x] \geq 0$ **then**
                Let z be the total matches induced by a_i
                    if added to a (a', c', g', u', x)-sequence
                $M[i, j, a, c, g, a_i] := \max(M[i, j, a, c, g, a_i], M[i-1, j-1, a', c', g', x] + z)$
          if $a_i \neq b_j$ **then**
            for each $x \in \{A, C, G, U\}$
              $M[i, j, a, c, g, x] := \max(M[i-1, j, a, c, g, x], M[i, j-1, a, c, g, x])$

Theorem 4 *The* **LCSBM** *problem can be solved in* $O(n^{m+3})$ *time. When* $m = 2$*, the problem can be solved in* $O(n^5)$ *time.*

5 Concluding Remarks

In this paper, we study several versions of the problem for RNA multiple structural alignment using a LCS model. Under this new model, we can solve a list of fundamental problems in polynomial time. We hope that these results and some similar ideas can be incorporated with known methods to enhance research in RNA multiple structural alignment. There are several interesting open problems related to this work: (1) Can the bounds in this paper be further improved? In RNA based biological applications, n could be around 1000. In that case, reducing the running time is important. (2) In [5], given a multiple number of linear graphs, each with n vertices, computing the maximum common non-intersecting subgraph was shown to be NP-complete. But the best $O(\log n)$ approximation factor in [13] is too high to make the result practically meaningful. Can it be further reduced?

References

[1] F. Y. L. Chin, A. De Santis, A. L. Ferrara, N. L. Ho, S. K. Kim. A simple algorithm for the constrained sequence problems. *Inform. Proc. Letters*, **90(4)**:175-179, 2004.

[2] T. Cormen, C. Leiserson, R. Rivest, C. Stein. *Introduction to Algorithms*, second edition, MIT Press, 2001.

[3] M. Dayhoff. Computer aids to protein sequence determination. *J. Theoret. Biology*, **8(1)**:97-112, 1965.

[4] M. Dayhoff. Computer analysis of protein evolution. *Scientific American*, **221(1)**:86-95, 1969.

[5] E. Davydov and S. Batzoglu. A computational model for RNA multiple structural alignment. *Proc. 15th Ann. Symp. Combinatorial Pattern Matching,* LNCS 3109, pp. 254-269, 2004.

[6] X. Deng, G. Li, Z. Li, B. Ma and L. Wang. A PTAS for distinguishing (sub)string selection. *Proc. ICALP'02*, pp. 740-751, 2002.

[7] S.R. Eddy. Noncoding RNA genes and the modern RNA world. *Nature review Genetics*, **2:**919-929, 2001.

[8] D. Goldman, S. Istrail and C. Papadimitriou. Algorithmic aspects of protein structure similarity. *Proc. 40th Ann. Symp. Foundations of Computer Science (FOCS'99)*, pp. 512-522, 1999.

[9] R. I. Greenberg. Bounds on the Number of the Longest Common Subsequence Problem. *CoRR cs.DM/0301030*, 2003.

[10] D. Hirschberg. The longest common subsequence problem. *PhD Thesis*, Princeton University, 1975.

[11] W.J. Hsu and M.W. Du. Computing a longest common subsequence for a set of strings. *BIT*, **24:**45-59, 1984.

[12] T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM J. Comput.*, **24(5)**:1122-1139, 1995.

[13] M. Kubica, R. Rizzi, S. Vialette and T. Walen. Approximation of RNA multiple structural alignment. *Proc. 17th Ann. Symp. Combinatorial Pattern Matching,* LNCS 4009, pp. 211-222, 2006.

[14] K. Lanctot, M. Li, B. Ma, S. Wang and L. Zhang. Distinguishing string selection problems. *Proc. 6th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pp. 633-642, 1999.

[15] M. Li, B. Ma and L. Wang. Finding similar regions in many strings. *Proc. 31st ACM Symp. on Theory of Computing (STOC'99)*, pp. 473-482, 1999.

[16] D. Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, **25:**322-336, 1978.

[17] R. Nussinov, G. Pieczenik, J. Griggs and D. Kleitman. Algorithms for loop matching. *SIAM J. Applied Math.*, **35:**68-82, 1978.

[18] C. Rick. Efficient Computation of All Longest Common Subsequences. *Proc. 7th Scandinavian Workshop on Algorithm Theory (SWAT'00)*, pp. 407-418, 2000.

[19] E. Rivas and S.R. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *J. Molecular Biology*, **285:**2053-2068, 1999.

[20] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J. Molecular Biology*, **147:**195-197, 1981.

[21] Y.-T. Tsai, The constrained longest common subsequence problem, *Inform. Proc. Letters*, **88(4)**:173-176, 2003.

[22] M. Zucker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, **9:**133-148, 1981.

[23] M. Zucker. Computer prediction of RNA structure. *Methods in Enzymology*, **180:**262-288, 1989.