A Faster and More Space-Efficient Algorithm for Inferring Arc-Annotations of RNA Sequences Through Alignment

Jesper Jansson[1], See-Kiong Ng[2], Wing-Kin Sung[1], and Hugo Willy[1]

[1] Department of Computer Science, National University of Singapore
3 Science Drive 2, Singapore 117543
{jansson,ksung,hugowill}@comp.nus.edu.sg
[2] Institute for Infocomm Research, 21 Heng Mui Keng Terrace, Singapore
skng@i2r.a-star.edu.sg

Abstract. This paper considers the problem of inferring the optimal nested arc-annotation of a sequence given another nested arc-annotated sequence by maximizing the weighted alignment between the bases and arcs in the two sequences. The problem has a direct application in predicting the secondary structure of an RNA sequence given a closely related sequence whose secondary structure is already known. The currently most efficient algorithm for this problem requires $O(nm^3)$ time and $O(nm^2)$ space where n is the length of the sequence with known arc-annotation while m is the length of the sequence to be inferred. We present an improved algorithm which runs in $\min\{O(nm^2 \log n), O(nm^3)\}$ time and $\min\{O(m^2 + mn), O(m^2 \log n)\}$ space. The time improvement is achieved by applying sparsification to the dynamic programming algorithm, while the space is reduced to a more practical quadratic complexity by using a Hirschberg-like traceback technique together with a simple compression.

1 Introduction

Recent research shows that RNA functions as catalysts and regulators in nucleic acid processing and gene expression in addition to its commonly known intermediary role in DNA transcription and translation process. It is generally known that much of RNA's functionalities depend on its structural features. Unfortunately, although massive amount of sequence data are continuously generated, the number of known RNA structures is still very limited since experimental methods, such as NMR and Crystallography, require expertise and long experimental time. Therefore, computational methods for predicting RNA structure are very useful.

There exist a number of computational approaches to predict the structure of RNA in the literature. Basically, they can be classified into three categories: Energy Minimization, Comparative, and Structure Inferring methods. The first approach tries to compute the structure of an RNA molecule which has the lowest free energy. Representatives of this approach are the methods of Nussinov et al [16] and Zuker et al [15, 20, 21]. Since the current energy model is not

I. Jonassen and J. Kim (Eds.): WABI 2004, LNBI 3240, pp. 302–313, 2004.

accurate enough and RNA may not fold into the lowest energy structure, the prediction accuracy of this method is usually not high. For the Comparative method, we are given a number of RNA sequences which are expected to have similar structure called the homologous sequences. By aligning those RNA sequences, we compute the consensus structure. Representatives of this approach include Maximum Weighted Matching (MWM) [3, 18] and Stochastic Context Free Grammars (SCFGs) [8, 7, 17]. The Comparative approach is currently the best way to predict RNA structures [9, 12]. However, when the number of homologous sequences is not large enough, the accuracy can be low. If we only have a few homologous RNA sequences where the structure of one of the sequences is known, the RNA structure can be predicted using the Structure Inferring method [2, 19]. Consider two sequences S_1 and S_2 of length n and m. Assuming that the secondary structure of S_1 is known, this method infers the secondary structure of S_2 by aligning S_1 and S_2. Bafna et al [2] propose a dynamic programming solution to this problem and solve it using $O(n^2m^2 + nm^3)$ time and $O(n^2m^2)$ space. Zhang [19] improves their result and gives an algorithm which runs in $O(nm^3)$ time and $O(nm^2)$ space. In this paper, we further improve the running time of the inference algorithm to $\min\{O(nm^2 \log n), O(nm^3)\}$ and at the same time bring down the space requirement to $\min\{O(m^2 + mn), O(m^2 \log n)\}$.

Our improvement in the running time stems from sparsification. We observe that the entries in every row and every column in the dynamic programming tables are monotonically increasing, enabling us to calculate less entries in the tables without losing any information. We also designed a new recursive dynamic programming algorithm that gives a better worst-case space requirement in the case of computing only the score of the alignment of S_1 and S_2. Finally, by incorporating the latter into an algorithm similar to Hirschberg's traceback [10] together with a simple compression method, we can recover the optimal inferred structure from the table within the stated reduced space complexity. Note that the space improvement is critical in our application since currently the length of a typical RNA sequence used in lab experiments is around 3K to 5K bases. Assuming that $n \approx m$, the memory requirement of an $O(nm^2)$ space algorithm could easily reach over tens of gigabytes. This memory requirement is not impossible to meet but it is highly impractical.

This paper is organized as follows. Section 2 contains the formal statement of the problem with some basic definitions. Section 3 presents the algorithm and is divided into three parts. The first part presents the original algorithm given in [19], noting the bottleneck of the computation. The following two parts present our techniques to improve the running time of the algorithm. The Hirschberg-like traceback algorithm is described in Section 4. Finally, Section 5 concludes this paper with some possible extensions of the problem.

2 Preliminaries

We use a slightly different notation from the one in [19] where the secondary structure of the first sequence is represented as a tree. Each internal node in the

tree represents a base pair and the bases in the loop created by the base pair are the children of the node.

In our algorithm, we represent an RNA sequence and its secondary structure information using the *arc-annotated* sequence [4]. Consider a sequence S over a fixed alphabet $\Sigma = \{A, C, G, U\}$. We define $S[i]$ to be the i^{th} character in S and $S[i..j]$ to be the substring of S in positions between i and j (inclusive). For any $x \in \Sigma$, let $Complement(x)$ be the complementary base of x based on the Watson-Crick base pairing. For example $Complement(A)$ is U and $Complement(G)$ is C. An unordered pair of positions (i, j), where $i < j$, indicates that $S[i]$ and $S[j]$ form a base pair in the RNA structure. Such pair is called an *arc*. For RNA sequences, it is required that $S[j] = Complement(S[i])$ and vice versa. A set P of arcs is called an *arc-annotation*, and the pair (S, P) is called an *arc-annotated* sequence. Arc-annotated sequences are well-studied [1, 4, 6, 11, 13, 14, 19] and are commonly used in computational biology to represent the structure of RNA and protein sequences. Since we are considering RNA secondary structures, we assume that the RNA sequences we are dealing with do not have any pseudoknots. The corresponding arc-annotation construct for such RNA structures is the *nested* arc-annotation [1, 11, 13, 14] where, given two arcs, either one is within the other, or they are completely disjoint $(\forall (i_1, j_1), (i_2, j_2) \in P,\ i_1 \in [i_2, j_2] \Leftrightarrow\ j_1 \in [i_2, j_2])$. For any arc $u \in P$, we denote u_l and u_r to be the left and the right endpoints of u, respectively. The *size* of an arc u is denoted by $|u| = u_r - u_l + 1$. We say that position i is *free* if i is not an endpoint of any arc in P. A position i is *covered* by an arc u if $u_l < i < u_r$ and there exist no other arc u' such that $u'_l < i < u'_r$. The set of all positions covered by u is called the *arc cover* of u, denoted by $C(u)$.

Given two arc-annotated sequences (S_1, P_1) and (S_2, P_2), we can define the similarity of the sequences by aligning the bases and the arcs in them. We need to define a scoring function for each type of alignment. Let χ be the function to score the alignment of unpaired bases in the two sequences where, for $a, b \in \{A, C, G, U, \sqcup\}$, $\chi(a, b) = \beta$ if $a = b$ and 0 otherwise ('$\sqcup$' denotes a blank character). For any arc u, which represents paired bases in the RNA structure, let δ be a scoring function for arcs alignment whose value is defined as:

$$\delta((S_1[u_l], S_1[u_r]),(S_2[j], S_2[j'])) = \begin{cases} \alpha_1 & \text{if } S_1[u_l] = S_2[j] \text{ and } S_1[u_r] = S_2[j'] \\ \alpha_2 & \text{if } S_1[u_l] \neq S_2[j] \text{ and } S_1[u_r] \neq S_2[j'] \\ & \quad \text{but } S_2[j] = Complement(S_2[j']) \\ -\infty & \text{otherwise} \end{cases}$$

β, α_1, and α_2 are positive integer constants. Usually the parameters are set such that $\beta \leq \alpha_2 \leq \alpha_1$ which reflects that an arc-alignment(α_1 or α_2) takes precedence over single base alignment(β). Moreover, an arc alignment with exactly the same base pairs should score higher (α_1) since both bases and their arc are aligned. One can also have constraints on the arc width. For example, when $|j - j'|$ is less than some minimum arc width parameter, we can define $\delta = -\infty$. Now given the definition of the arc annotation and the scoring functions, we formally state our problem (slightly altered from the one in [19]) as follows.

The Weighted Largest Common Substructure(WLCS) of two arc annotated sequences (S_1, P_1) and (S_2, P_2) is defined as the maximum weighted alignment between S_1 and S_2 where free bases are aligned to free bases and arcs are aligned to arcs. The WLCS score is then defined as the sum of all bases and arcs alignment scores. The problem we address in this paper is: Given a nested arc-annotated sequence (S_1, P_1) and a plain sequence S_2, infer the nested arc-annotation P_2 for S_2 that maximizes their WLCS score.

3 Algorithm Description

This section reviews Zhang's algorithm (presented in [19]) for inferring the RNA secondary structure P_2 for S_2 that maximizes the WLCS score between (S_1, P_1) and (S_2, P_2). Let $|S_1| = n$ and $|S_2| = m$. Let $DP_{(i,i')}[j, j']$, where $1 \leq i \leq i' \leq n$ and $1 \leq j \leq j' \leq m$, denotes the score of the weighted largest common substructure between $(S_1[i..i'], P_1)$ and $S_2[j..j']$. Note that $DP_{(i,i')}[j, j'] = 0$ whenever $i > i'$ or $j > j'$. Zhang presented an algorithm which runs in $O(nm^3)$ time and uses $O(nm^2)$ space based on a two-step dynamic programming. Given an arc u, the first step computes the value of DP for the *arc-cover* of u i.e. computes $DP_{(u_l+1,u_r-1)}[j, j']$ for all $1 \leq j \leq j' \leq m$. Then, the next step computes the value of DP for the whole arc u, that is $DP_{(u_l,u_r)}[j, j']$ for all $1 \leq j \leq j' \leq m$. Below are the three equations in [19] to compute the two steps in the algorithm. Please refer to the paper for the proofs.

Lemma 1. *(Lemma 4 in [19]) If either i' is free or i' is an endpoint of an arc whose other endpoint is not in $[i..i'-1]$,*

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

Lemma 2. *(Lemma 5 in [19]) For any arc $u \in P_1$ and $i < u_l$,*

$$DP_{(i,u_r)}[j, j'] = \max_{j-1 \leq j'' \leq j'} \{DP_{(i,u_l-1)}[j, j''] + DP_{(u_l,u_r)}[j''+1, j']\}$$

Lemma 3. *(Lemma 3 in [19]) For any arc $u \in P_1$,*

$$DP_{(u_l,u_r)}[j, j'] = \max \begin{cases} DP_{(u_l+1,u_r-1)}[j+1, j'-1] + \\ \qquad \delta((S_1[u_l], S_1[u_r]), (S_2[j], S_2[j'])), \\ DP_{(u_l+1,u_r-1)}[j, j'], \\ DP_{(u_l,u_r)}[j+1, j'], \\ DP_{(u_l,u_r)}[j, j'-1] \end{cases}$$

Definition 1. *If i' is free or i' is a right endpoint of an arc whose left endpoint is not in $[i..i']$, then given the table $DP_{(i,i'-1)}$, $DP_{(i,i')}$ can be computed by using Lemma 1. We define the computation of $DP_{(i,i')}$ from $DP_{(i,i'-1)}$ as the operation $EXTEND(DP_{(i,i'-1)})$.*

$WLCS(S_1, P_1, P_2)$
For every arc $u \in P_1$ from the innermost to the outermost, left to right,

Step 1 : Compute $DP_{(u_l+1,u_r-1)}$ as follows.
For every $i \in C(u)$ in increasing order,
- if i is free, compute $DP_{(u_l+1,i)}$ by EXTEND($DP_{(u_l+1,i-1)}$).
- if $i = v_r$ for some arc v, compute $DP_{(u_l+1,i)}$ by MERGE($DP_{(u_l+1,v_l-1)}, DP_{(v_l,v_r)}$).
- if $i = v_l$, do nothing.

Step 2 : Compute $DP_{(u_l,u_r)}$ by ARC-MATCH($DP_{(u_l+1,u_r-1)}$).

Fig. 1. The algorithm from [19] described in terms of EXTEND, MERGE and ARC-MATCH operations.

Definition 2. *Consider any arc s. The operation MERGE($DP_{(i,s_l-1)}, DP_{(s_l,s_r)}$) is defined to be the computation of the table $DP_{(i,s_r)}$ given $DP_{(i,s_l-1)}$ and $DP_{(s_l,s_r)}$ using Lemma 2.*

Definition 3. *Consider any arc s. The operation ARC-MATCH($DP_{(s_l+1,s_r-1)}$) is defined to be the computation of the table $DP_{(s_l,s_r)}$ given $DP_{(s_l+1,s_r-1)}$ using Lemma 3.*

Fig. 1 describes the procedure $WLCS(S_1, P_1, S_2)$ which computes $DP_{(1,n)}[j, j']$ for all $1 \leq j \leq j' \leq m$ based on the algorithm in [19]. As analyzed in the latter, EXTEND takes $O(m^2)$ time. There are $O(n)$ free bases in S_1; thus, all calls to EXTEND require a total of $O(nm^2)$ time. The procedure MERGE will need to fill $O(m^2)$ entries in the combined table, each requires $O(m)$ time to compute because we need to find the maximum over $O(m)$ sums, in the worst case. Since MERGE is only invoked on arcs and the number of arcs in P_1 could reach $O(n)$; in total, all calls to MERGE require $O(nm^3)$ time. ARC-MATCH computes the term in Lemma 3 over $O(m^2)$ (j, j') pairs for each arc in P_1. Based on a similar argument on the number of arcs in P_1, ARC-MATCH requires $O(nm^2)$ time. As for the space requirement, assuming the standard traceback for inferring the secondary structure of the sequence S_2, we must store all intermediary DP tables computed by $WLCS(S_1, P_1, S_2)$. The cardinality of the latter is bounded by $O(n)$ as the number of free bases and arcs are both bounded by $O(n)$. In conclusion, the time and space complexity of the whole algorithm is $O(nm^3)$ and $O(nm^2)$, respectively.

3.1 The Sparsification Technique – Monotonically Increasing Property of DP

The previous section shows that the bottleneck of the computation of the WLCS score is in the procedure MERGE. Here, we describe how to speed up the computation of MERGE by taking advantage of the properties of $DP_{(i,i')}$.

Observation 1 *For any $i \leq i'$, $DP_{(i,i')}$ satisfies the following properties.*

1. *In every row j of $DP_{(i,i')}$, the entries are monotonically increasing, i.e.,* $DP_{(i,i')}[j,j'] \leq DP_{(i,i')}[j,j'+1]$.
2. *In every column j' of $DP_{(i,i')}$, the entries are monotonically decreasing, i.e.,* $DP_{(i,i')}[j,j'] \geq DP_{(i,i')}[j+1,j']$.

The observations above motivate the following definitions.

Definition 4. *[5] For every row j of $DP_{(i,i')}$, a position j^* satisfying $j \leq j^* \leq m$ is defined to be a* row interval point *if $DP_{(i,i')}[j,j^*-1] < DP_{(i,i')}[j,j^*]$. The set of row interval points j^* in the j^{th} row of $DP_{(i,i')}$ is denoted by* $\mathrm{RowIP}_j(DP_{(i,i')})$.

Definition 5. *[5] For every column j of $DP_{(i,i')}$, a position j^* satisfying $1 \leq j^* \leq j$ is defined to be a* column interval point *if $DP_{(i,i')}[j^*,j] > DP_{(i,i')}[j^*+1,j]$. The set of column interval points j^* in the j^{th} column of $DP_{(i,i')}$ is denoted by* $\mathrm{ColIP}_j(DP_{(i,i')})$.

Lemma 4. *Let $\alpha = \max\{\beta, \alpha_1, \alpha_2\}$. Then there are at most* $(\min\{\alpha(i'-i+1), (m-j+1)\})$ *row interval points in any row j of $DP_{(i,i')}$.*

Proof. (Sketch) The total number of row interval points (which are all distinct) in any row j of $DP_{(i,i')}$ is bounded by the minimum of the maximum (integer) score and the number of columns in the row. □

Corollary 1. *There are at most* $(\min\{\alpha(i'-i+1), j'\})$ *column interval points in any column j' of $DP_{(i,i')}$.*

In [19], for every (j,j') pair where $j \leq j'$, the procedure MERGE($DP_{(i,i')}$, $DP_{(i'+1,i'')}$) tries every possible $j'' \in [j-1..j']$ to compute the one that maximizes

$$DP_{(i,i')}[j,j''] + DP_{(i'+1,i'')}[j''+1,j'] \tag{1}$$

The following lemma states that it is unnecessary to consider all $j'' \in [j-1..j']$ to find the maximum of (1).

Lemma 5. *The equation from Lemma 2 can be computed by*

$$DP_{(i,u_r)}[j,j'] =$$

$$\max_{\substack{j^* \in \mathrm{RowIP}_{j(DP_{(i,u_l-1)}) \cup \{j-1\}} \\ j^* \leq j'}} \{DP_{(i,u_l-1)}[j,j^*] + DP_{(u_l,u_r)}[j^*+1,j']\}$$

which checks at most $(\min\{\alpha(u_l-i)+1, (j'-j+1)\})$ *candidates of j^*.*

Proof. (Sketch) Let $F(j'') = DP_{(i,i')}[j,j''] + DP_{(i'+1,i'')}[j''+1,j']$. By Lemma 2, $DP_{(i,u_r)}[j,j'] = \max\{F(j-1), \max_{j'' \in [j..j']} F[j'']\}$. For each $j'' \in [j..j']$, we observe that there exists a $j^* \in \mathrm{RowIP}_j(DP_{(i,u_l-1)})$ such that $j^* \leq j''$ and $DP_{(i,u_l-1)}[j,j^*] = DP_{(i,u_l-1)}[j,j'']$. Furthermore, since $j^* \leq j''$, we have $DP_{(u_l,u_r)}[j^*+1,j'] \geq DP_{(u_l,u_r)}[j''+1,j']$. Hence, for such j^* we have $F[j^*] \geq F[j'']$ resulting in $\max_{j'' \in [j..j']} F[j''] = \max_{j^* \in \mathrm{RowIP}_{j(DP_{(i,u_l-1)})}} F[j^*]$. □

Corollary 2. *The equation from Lemma 2 can be computed by*

$$DP_{(i,u_r)}[j,j'] =$$

$$\max_{\substack{j^*+1 \in \mathrm{ColIP}_{j'(DP_{(u_l,u_r)})\cup\{j'+1\}} \\ j^*+1 \geq j}} \{DP_{(i,u_l-1)}[j,j^*] + DP_{(u_l,u_r)}[j^*+1,j']\}$$

which checks at most $(\min\{\alpha|u|+1,(j'-j+1)\})$ *candidates of* j^*.

By Lemma 5 and Corollary 2, the time complexity of MERGE($DP_{(i,i'-1)}$, $DP_{(i',i'')}$) is improved to $O(\min\{\alpha(i''-i')m^2, \alpha(i'-i)m^2, m^3\})$.

3.2 The Recursive Dynamic Programming Algorithm

We now introduce a new algorithm $WLCS_r(S_1, P_1, S_2)$ which computes the table $DP_{(1,n)}$ using a carefully designed recursive dynamic programming algorithm. This improved algorithm guarantees that each MERGE operation is applied only on arcs whose size is at most half of its parent's[1].

Let us start with some definitions. The followings are with respect to a *nested* annotated structure. An arc u is a *parent* of an arc v (denoted by $Parent(v)$) if $u_l < v_l < v_r < u_r$ and there is no arc w such that $u_l < w_l < v_l < v_r < w_r < u_r$. Conversely, v is referred as the *child* of the arc u. The set of children of an arc u is denoted by $Child(u)$. A *core-arc*, with respect to an arc u, is a child of u which has the biggest size (denoted as core-arc(u)). All other children of u are named *side-arcs* and form the set *side-arcs*(u). A *terminal-arc* is defined to be an arc which has no child. For any arc $u \in P_1$, the *core-path* $CP(u)$ is an ordered set of core-arcs $\{c_1, c_2, \cdots, c_\ell\}$, where $c_1 = u$ and for any c_i, c_{i+1} is core-arc(c_i)

$WLCS_r(S_1, S_2)$ first finds the largest arc u in $[1..n]$ and processes every core-arc $c \in CP(u)$ from the innermost to the outermost. For terminal arcs t, $DP_{(t_l,t_r)}$ can be computed by using EXTEND operations only. For the remaining arcs c, $DP_{(c_l,c_r)}$ is obtained using a *two-part computation*. Let c' be core-arc(c). Due to the bottom-up ordering, $DP_{(c'_l,c'_r)}$ will have been computed at this point of time. We first compute the value of $DP_{(c_l+1,c'_l-1)}$ (the *LEFT Part* phase) using EXTEND and MERGE operations. Given $DP_{(c_l+1,c'_l-1)}$, we proceed using EXTEND and MERGE to compute $DP_{(c'_l,c_r-1)}$(the *RIGHT Part* phase). In both phases, whenever we encounter a side-arc s, we first compute $DP_{(s_l,s_r)}$ by recursively calling $WLCS_r(S_1[s_l..s_r], S_2)$. Next, we apply MERGE on $DP_{(c_l+1,c'_l-1)}$ and $DP_{(c'_l,c_r-1)}$ to compute $DP_{(c_l+1,c_r-1)}$. Finally, $DP_{(c_l,c_r)}$ is obtained by applying ARC-MATCH($DP_{(c_l+1,c_r-1)}$). If $(1,n) \in P_1$, then $u = (1,n)$ and we are done. Otherwise, we need to compute $DP_{(1,n)}$ using the same two-part

[1] The routine $WLCS(S_1, P_1, P_2)$ given in [19] computes the DP tables according to the postorder of the nodes in their tree representation. The problem of this approach is that we may need to perform MERGE on arcs with large sizes causing an $\Omega(nm^2)$ space requirement even if we only wish to compute the WLCS score of (S_1, P_1) and S_2. We shall prove this claim in the full version of this paper.

WLCS$_r(S_1, P_1, S_2)$ /* $|S_1| = n, |S_2| = m$ */

- Let u be biggest arc in P_1 and $CoreArcs = CP(u)$.
- Let $DP_{(i,i')}$ be the $m \times m$ score matrix of $DP_{(i,i')}[j, j']$, $1 \le j \le j' \le m$.
- For the terminal-arc $t \in CoreArcs$,
  - For $k = t_l + 1$ to $t_r - 1$
    - ◇ Compute $DP_{(t_l+1,k)}$ using EXTEND($DP_{(t_l+1,k-1)}$).
  - Compute $DP_{(t_l,t_r)}$ by ARC-MATCH($DP_{(t_l+1,t_r-1)}$).
- For every core-arc $c \in CoreArcs$ in bottom-up order, $c' =$ core-arc(c)
  - **LEFT Part**
    For $k = c_l + 1$ to $c'_l - 1$ where $k \in C(c)$,
    - ○ If k is free,
      - ◇ Compute $DP_{(c_l+1,k)}$ using EXTEND($DP_{(c_l+1,k-1)}$).
    - ○ If $k = s_r$ for some $s \in$ side-arcs(c),
      - ◇ Compute $DP_{(s_l,s_r)}$ recursively by $WLCS_r(S_1[s_l..s_r], P_1, S_2)$.
      - ◇ Compute $DP_{(c_l+1,k)}$ using MERGE($DP_{(c_l+1,s_l-1)}, DP_{(s_l,s_r)}$).
  - **RIGHT Part**
    For $k = c'_r + 1$ to $c_r - 1$ where $k \in C(c)$,
    - ○ If k is free,
      - ◇ Compute $DP_{(c'_l,k)}$ using EXTEND($DP_{(c'_l,k-1)}$).
    - ○ If $k = s_r$ for some $s \in$ side-arcs(c),
      - ◇ Compute $DP_{(s_l,s_r)}$ recursively by $WLCS_r(S_1[s_l..s_r], P_1, S_2)$.
      - ◇ Compute $DP_{(c'_l,k)}$ using MERGE($DP_{(c'_l,s_l-1)}, DP_{(s_l,s_r)}$).
  - Compute $DP_{(c_l+1,c_r-1)}$ by MERGE($DP_{(c_l+1,c'_l-1)}, DP_{(c'_l,c_r-1)}$).
  - Compute $DP_{(c_l,c_r)}$ using ARC-MATCH($DP_{(c_l+1,c_r-1)}$).
- If $u \neq (1, n)$
  - Compute $DP_{(1,u_l-1)}$ by the **LEFT Part** computation.
  - Compute $DP_{(u_l,n)}$ by the **RIGHT Part** computation.
  - Compute $DP_{(1,n)}$ by MERGE($DP_{(1,u_l-1)}, DP_{(u_l,n)}$).

Fig. 2. The algorithm $WLCS_r(S_1, P_1, S_2)$.

computation technique: first compute $DP_{(1,u_l-1)}$, followed by $DP_{(u_l,n)}$, and then obtain $DP_{(1,n)}$ by MERGE($DP_{(1,u_l-1)}, DP_{(u_l,n)}$). Our complete algorithm $WLCS_r(S_1, P_1, S_2)$ is listed in Fig. 2.

Lemma 6. *$WLCS_r(S_1, P_1, S_2)$ runs in $\min\{O(\alpha nm^2 \log n), O(nm^3)\}$ time.*

Proof. To obtain the execution time of $WLCS_r(S_1, P_1, S_2)$, we analyze the total execution time of the EXTEND, MERGE, and ARC-MATCH operations separately. The time required by EXTEND and ARC-MATCH operations is still the same as in [19], namely $O(nm^2)$, as they are still applied at most once on every free bases and arcs, respectively. Note that MERGE is now invoked on all arcs which belong to the set side-arc(u) for some arc $u \in P_1$ and on the merging of the LEFT part and the RIGHT part of all non-terminal arcs. For any side-arc s, merging the table $DP_{(s_l,s_r)}$ into some table $DP_{(i,s_l-1)}$ takes at most $O(\min\{\alpha|s|m^2, \alpha(s_l - i)m^2, m^3\})$ which is at most $O(\min\{\alpha|s|m^2, m^3\})$ time. In the second type of MERGE invocations, we execute MERGE($DP_{(c_l+1,c'_l-1)}$,

$DP_{(c'_l,c_r-1)}$) for all non-terminal arcs c where c' =core-arc(c). The latter requires $O(\min\{\alpha(c'_l - c_l)m^2, \alpha(c_r - c'_l)m^2, m^3\}) \leq O(\min\{\alpha(c'_l - c_l)m^2, m^3\})$. Let r be an imaginary arc where $r = (0, n+1)$ and $T(S_1)$ be the total execution time for all MERGE operations in $WLCS_r(S_1, P_1, S_2)$.

$$T(S_1) = \sum_{c \in CP(r)} \Big(\sum_{s \in \text{side-arcs}(c)} T(S_1[s_l..s_r]) + O(\min\{\alpha|s|m^2, m^3\}\Big) + \sum_{c \in CP(r)} O(\min\{\alpha(c'_l - c_l)m^2, m^3\}) \quad (2)$$

$$= \sum_{\substack{s \in \text{side-arcs}(c) \\ c \in CP(r)}} T(S_1[s_l..s_r]) + \sum_{\substack{s \in \text{side-arcs}(c) \\ c \in CP(r)}} O(\min\{\alpha|s|m^2, m^3\}) + \sum_{c \in CP(r)} O(\min\{\alpha(c'_l - c_l)m^2, m^3\}) \quad (3)$$

$$= \sum_{\substack{s \in \text{side-arcs}(c) \\ c \in CP(r)}} T(S_1[s_l..s_r]) + O(\min\{\alpha nm^2, m^3\}) \quad (4)$$

Both the second and the third summation terms in (3) sum up to $O(\min\{\alpha nm^2, m^3\})$ since all side-arcs $s \in$ side-arcs(c) as well as the ranges $[c'_l..c_l]$ for $c \in CP(r)$ are non overlapping. Based on the fact that $\sum |s| \leq |c|$ and $|s| \leq \frac{|c|}{2}$, by inspection, the solution of the recurrence is $O(\alpha nm^2 \log n)$ if $\min\{\alpha nm^2, m^3\} = \alpha nm^2$ or $O(nm^3)$ otherwise. Combining the running time of the three operations, the lemma follows. □

4 Traceback Using a Hirschberg-Based Technique

Using the standard traceback algorithm, one is required to store all DP tables corresponding to any arc $u \in P_1$. Alternatively, we can make use of the recursive technique introduced by Hirschberg in [10] and use $WLCS_r(S_1, P_1, S_2)$ only to compute the WLCS score. We shall refer to the latter as the *score-only* $WLCS_r(S_1, P_1, S_2)$.

Lemma 7. *Computing the score-only* $WLCS_r(S_1, P_1, S_2)$ *requires* $\min\{O(m^2 \log n), O(m^2 + \alpha mn)\}$ *space.*

Proof. To compute the score-only $WLCS_r(S_1, P_1, S_2)$, since we do not have to traceback, we can just store the information needed to compute the alignment score. This corresponds to $O(m^2)$ space for the EXTEND and ARC-MATCH operations. As for the MERGE operations, when there is no recursive call involved (the second type of MERGE), the space requirement is also in $O(m^2)$. Otherwise, referring back to Fig. 2, when we invoke the recursive call $WLCS_r(S_1[s_l..s_r], P_1, S_2)$, we observe that we need to store $DP_{(i,s_l-1)}$ for some fixed i. Storing only the row interval points takes $O(\min\{\alpha(s_l - i)m, m^2\})$ space (by Lemma 4). Since recursive calls are only applied on side-arcs, we have at

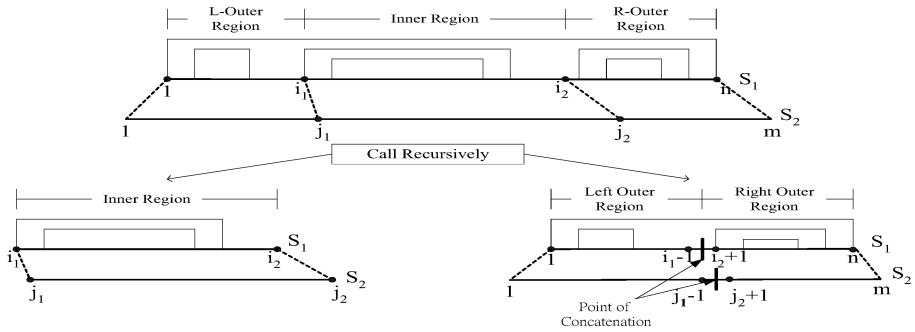

Fig. 3. The recursion on the partitioned continuous region by Lemma 8.

most $O(\log n)$ recursion levels giving an upper bound of $O(m^2 \log n)$ on the space complexity. We further claim that the space required is smaller than $O(\alpha nm)$ since, in each recursion level x, we only store $DP_{(i_x, s_{l_x}-1)}$ where all of the intervals $[i_x..s_{l_x}-1]$ are disjoint. Hence, $\sum_x O(\alpha(s_{l_x} - i_x)m) \leq O(\alpha nm)$. Combining the three terms, the lemma follows. □

Following the idea of Hirschberg in [10], we compute the WLCS alignment between (S_1, P_1) and S_2 as follows,

1. Divide S_1 into a constant number of non-overlapping regions $S_{11}, S_{12}, ..S_{1c}$.
2. For each region S_{1i}, find the region S_{2i} in S_2 such that the optimal WLCS alignment will align S_{1i} to S_{2i}.
3. Recursively compute the optimal WLCS alignments between S_{1i} and S_{2i} for $i = 1, 2, .., c$.

To do the first step, since S_1 is arc-annotated, we must divide S_1 in such a way that we do not break any arc in P_1. The solution is to divide S_1 into *inner* and *outer* regions. Given two points i_1 and i_2, $1 \leq i_1 \leq i_2 \leq n$, the *inner* region with respect to i_1 and i_2 is $S_1[i_1..i_2]$ and the *outer* region is the concatenation of $S_1[1..(i_1 - 1)]$ and $S_1[(i_2 + 1)..n]$ (see Fig. 3). The latter is also referred as a *gapped* region since it has a discontinuous interval ($S_1[i_1..i_2]$ is removed). Let $\star$ be a special character that denotes the gap in the sequence such that the gapped region can be written as $S_1[1..(i_1 - 1)] \star S_1[(i_2 + 1)..n]$. If a region has no gap in it, we say it is *continuous*. We shall show that we can bound the size of each region by ϕn for some constant ϕ, $0 < \phi < 1$. Due to space constraints, the proofs of the following lemmas will appear in the journal version of this paper.

Lemma 8. *We can always partition a continuous region into* 2 *non-overlapping subregions, where one of them is continuous and the other is gapped. Every subregion's size is at most* $\frac{2}{3}$ *of the original region.*

Lemma 9. *We can always partition a gapped region into at most* 4 *non-overlapping subregions, where at most one of them is continuous. Every subregion's size is at most* $\frac{2}{3}$ *of the original region.*

After dividing S_1 into at most 4 subregions, where each is denoted by S_{1i} for $i \leq 4$, we now need to compute the regions S_{2i} in S_2 to which the subregions S_{1i} is aligned by the optimal WLCS alignment.

Lemma 10. *For any $1 \leq i_1 \leq i_2 \leq n$, we can compute $1 \leq j_1 \leq j_2 \leq m$, such that the optimal alignment between (S_1, P_1) and S_2 aligns $S_1[i_1..i_2]$ to $S_2[j_1..j_2]$, within the same time and space complexity of the score-only $WLCS_r(S_1, P_1, S_2)$.*

Lemma 11. *For any $1 \leq i_1 \leq i_2 \leq i_3 \leq i_4 \leq n$, we can compute $1 \leq j_1 \leq j_2 \leq j_3 \leq j_4 \leq m$, such that the optimal alignment between (S_1, P_1) and S_2 aligns $S_1[i_1..i_2] \star S_1[i_3..i_4]$ to $S_2[j_1..j_2] \star S_2[j_3..j_4]$, within the same time and space complexity of the score-only $WLCS_r(S_1, P_1, S_2)$.*

By Lemmas 10 and 11, the second step of this algorithm can be executed within the same time and space complexity of the score-only $WLCS_r(S_1, P_1, S_2)$ since the number of the subregions is constant. Next, we proceed by applying the algorithm recursively on each pair (S_{1i}, S_{2i}). While applying the algorithm on the continuous region is straightforward, the gapped region needs a bit of extra care. In this case, $\star$ in S_{1i} must be aligned to $\star$ in S_{2i} because they represent the subregion pair(s) computed in the other recursive call(s). To implement such constraint, we add into the base scoring function the following cases: $\chi(\star, \star) = 0$ and $\chi(\star, x) = \chi(x, \star) = -\infty$ for $x \in \{A, C, G, U, \sqcup\}$. This way, the optimal alignment between the two sequences is forced to align $\star$ in the first sequence to $\star$ in the second in order to have a non-negative score.

Lemma 12. *Our new algorithm can recover the optimal WLCS alignment in $\min\{O(\alpha nm^2 \log n), O(nm^3)\}$ time and $\min\{O(m^2 \log n)\ , O(m^2 + \alpha mn)\}$ space.*

5 Concluding Remarks

Consider two homologous RNA sequences S_1 and S_2 where S_1 has a known structure. This paper presents an improved algorithm to solve the problem of inferring the structure of S_2 such that the WLCS score between the two structures are maximized. The same algorithm can easily be applied to *the longest arc-preserving common subsequence problem* (LAPCS) (see, e.g., [4, 11]). In particular, we improve the time and space complexity of LAPCS (nested, plain) problem from $O(nm^3)$ and $O(nm^2)$[11] to $\min\{O(nm^2 \log n),\ O(nm^3)\}$ and $\min\{O(m^2 + mn),\ O(m^2 \log n)\}$.

One interesting extension of the problem discussed in this paper is to incorporate a more realistic, non-linear scoring function into the base and arc matching function. Another possible direction is to attempt some special cases of *crossed* arc-annotation structures, which can represent pseudoknotted structures in RNA sequences, by applying the algorithm iteratively.

References

1. J. Alber, J. Gramm, J. Guo, and R. Niedermeier. Towards optimally solving the longest common subsequence problem for sequences with nested arc annotations in linear time. In *CPM*, pages 99–114, 2002.

2. V. Bafna, S. Muthukrishnan, and R. Ravi. Computing similarity between RNA strings. In *CPM*, volume 937, pages 1–16, 1995.
3. R.B. Carey and G.D. Stormo. Graph-theoretic approach to RNA modeling using comparative data. In *ISMB*, pages 75–80, 1995.
4. P. A. Evans. *Algorithms and Complexity for Annotated Sequence Analysis*. PhD Thesis, University of Victoria, 1999.
5. W. Fu, W. K. Hon, and W. K. Sung. On all-substrings alignment problems. In *COCOON*, volume 2697, pages 80–89, 2003.
6. J. Gramm, J. Guo, and R. Niedermeier. Pattern matching for arc-annotated sequences. In *FSTTCS*, volume 2556, pages 182–193, 2002.
7. L. Grate, M. Herbster, R. Hughey, I. S. Mian, H. Noller, and D. Haussler. RNA modeling using Gibbs sampling and stochastic context free grammars. In *ISMB*, pages 138–146, 1994.
8. L. Grate. Automatic RNA secondary structure determination with stochastic context-free grammars. In *ISMB*, pages 136–144, 1995.
9. R.R. Gutell, N. Larsen, and C.R. Woese. Lessons from an evolving rRNA: 16S and 23S rRNA structures from a comparative perspective. *Microbiological Reviews*, 58(1):10–26, 1994.
10. D. S. Hirschberg. Algorithms for the longest common subsequence problem. *J. Association of Computing Machinery*, 24(4):664–675, 1977.
11. T. Jiang, G. H. Lin, B. Ma, and K. Zhang. The longest common subsequence problem for arc-annotated sequences. In *CPM*, volume 1848, pages 154–165, 2000. To appear in Journal of Discrete Algorithms.
12. D.A.M. Konings and R.R. Gutell. A comparison of thermodynamic foldings with comparatively derived structures of 16s and 16s-like rRNAs. *RNA*, 1:559–574, 1995.
13. G. H. Lin, Z. Z. Chen, T. Jiang, and J. Wen. The longest common subsequence problem for sequences with nested arc annotation. *Journal of Computer and System Sciences*, 65:465–480, 2002.
14. G. H. Lin, B. Ma, and K. Zhang. Edit distance between two RNA structures. In *RECOMB*, pages 211–200, 2001.
15. R. B. Lyngsø, M. Zuker, and C.N.S. Pedersen. Internal loops in RNA secondary structure prediction. In *RECOMB*, pages 260–267, 1999.
16. R. Nussinov and A.B. Jacobson. Fast algorithm for predicting the secondary structure of single stranded RNA. In *PNAS*, volume 77(11), pages 6309–6313, 1980.
17. Y. Sakakibara, M. Brown, R. Hughey, I.S. Mian, K. Sjölander, R.C. Underwood, and D. Haussler. Recent methods for RNA modeling using stochastic context-free grammars. In *Proc. of the Asilomar Conference on Combinatorial Pattern Matching*, 1994.
18. J.E. Tabaska, H.N. Gabow R.B. Cary, and G.D. Stormo. An RNA folding method capable of identifying pseudoknots and base triples. *Bioinformatics*, 14(8):691–699, 1998.
19. K. Zhang. Computing similarity between RNA secondary structures. In *IEEE International Joint Symposia on Intelligence and Systems*, pages 126–132, 1998.
20. M. Zuker. Prediction of RNA secondary structure by energy minimization. In *Methods in Molecular Biology*, volume 25, pages 267–94, 1994.
21. M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acid Res. 9*, pages 133–148, 1981.