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

Jesper Jansson † See-Kiong Ng * Wing-Kin Sung †‡ Hugo Willy†

Abstract

The nested arc-annotation of a sequence is an important model used to represent structural information for RNA and protein sequences. Given two sequences S_1 and S_2 and a nested arc-annotation P_1 for S_1, this paper considers the problem of inferring the nested arc-annotation P_2 for S_2 such that (S_1, P_1) and (S_2, P_2) have the largest common substructure. The problem has a direct application in predicting the secondary structure of an RNA sequence given a closely related sequence with known secondary structure. 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 and m is the length of the sequence whose arc-annotation is to be inferred. By using sparsification on a new recursive dynamic programming algorithm and applying a Hirschberg-like traceback technique with compression, we obtain an improved algorithm that runs in $\min\{O(nm^2 + n^2m), O(nm^2 \log n), O(nm^3)\}$ time and $\min\{O(m^2 + mn), O(m^2 \log n + n)\}$ space.

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 amounts 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.

In the literature, there exist a number of computational approaches to predict the structure of RNA. 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 with the lowest free energy. Representatives of this approach are the methods of Nussinov et al [17] and Zuker et al [16, 21, 22]. Since the current energy model is not 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 that are expected to share a similar structure(called homologous sequences). By aligning those RNA sequences, we can compute the consensus structure. Representatives of this approach include Maximum Weighted Matching (MWM) [3, 19] and Stochastic Context Free Grammars (SCFGs) [8, 9, 18]. The Comparative approach is currently the best way to predict RNA structures [10, 13]. However, when the number of homologous sequences is not

*Institute for Infocomm Research, 21 Heng Mui Keng Terrace, Singapore 119613, skng@i2r.a-star.edu.sg

†Department of Computer Science, National University of Singapore, 3 Science Drive 2, Singapore 117543, jansson@comp.nus.edu.sg, ksung@comp.nus.edu.sg, hugowill@comp.nus.edu.sg

‡Genome Institute of Singapore, Genome #02-01, 60 Biopolis Street Singapore 138672, sungk@gis.a-star.edu.sg

large enough, the accuracy can be low. If we only have a few homologous RNA sequences (which is usually the case) where the structure of one of the sequences is known, the RNA structure can be predicted using the Structure Inferring method [2, 20]. Consider two sequences S_1 and S_2 of length n and m, respectively. 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.

The formal definition of the problem is given in Section 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 [20] improves their result and gives an algorithm that runs in $O(nm^3)$ time and $O(nm^2)$ space. In this paper, we further improve the running time to $\min\{O(nm^2 + n^2m), O(nm^2 \log n), O(nm^3)\}$ and at the same time bring down the space requirement to min $\{O(m^2 + mn), O(m^2 \log n + n)\}$.

Our algorithmic improvement in the running time stems from a sparsification technique. We observe that the entries in every row in the dynamic programming tables are monotonically increasing, enabling us to fill in a smaller number of entries in the tables without losing any information. We also present a new recursive dynamic programming algorithm that gives a better worst-case space requirement for the case of computing only the score of the optimal alignment of S_1 and S_2. Finally, by incorporating the latter into an algorithm similar to Hirschberg's traceback [11] 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 RNA sequences used in lab experiments can reach thousands of 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 [20] in a slightly different notation, noting the bottleneck of the computation. The remaining two parts present our main results that improve the running time of the algorithm. Section 4 discusses a recursive dynamic programming algorithm that has a better space complexity compared to the original algorithm in [20] in the case where only the score of the alignment is required. The Hirschberg-like traceback algorithm is described in Section 5, making use of the score-only recursive algorithm described in the preceding section. Finally, Section 6 concludes this paper with some comments and possible future extensions of the problem.

2 Preliminaries

In our algorithm, we represent an RNA sequence and its secondary structure information using the arc-annotated sequence [4]. Let $[a..b]$ represents a discrete interval bounded by the integers a and b where $a \leq b$. When $a = b$, the interval can be written as $[a]$. 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(s) of x according to the Watson-Crick or Wobble (G-U) base pairing. Therefore, $Complement(A) = \{U\}$, $Complement(C) = \{G\}$, $Complement(U) = \{A, G\}$, and $Complement(G)$ is $\{C, U\}$. 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 a pair is called an *arc*. For RNA sequences, we require that, for any (i, j), $S[j] \in Complement(S[i])$ 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, 5, 7, 12, 14, 15, 20] 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 type of arc-annotation for RNA structures without pseudoknots is the *nested* arc-annotation [1, 12, 14, 15] where, for any 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 exists no other arc u' such that $u_l < u'_l < i < u'_r < u_r$. The set of all positions covered by u is called the *arc cover* of u, denoted by $C(u)$.

Consider two arc-annotated sequences (S_1, P_1) and (S_2, P_2). Let $|S_1| = n$ and $|S_2| = m$ where S_2 is the plain sequence whose arc-annotation P_2 is to be inferred. Given two arc-annotated sequences, 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 $a, b \in \{A, C, G, U, \sqcup\}$ ('$\sqcup$' denotes a blank character),

$$\chi(a, b) = \begin{cases} \beta & \text{if } a = b,\ a \neq \sqcup,\ b \neq \sqcup \\ 0 & \text{otherwise} \end{cases}$$

For any pair of position (u_1, u_2) in S_1 and (v_1, v_2) in S_2, let δ be a scoring function for arc alignment whose value is defined as:

$$\delta((S_1[u_1], S_1[u_2]), (S_2[v_1], S_2[v_2])) = \begin{cases} -\infty & \text{if } S_1[u_1] \notin \text{Complement}(S_1[u_2]) \text{ or} \\ & \quad S_2[v_1] \notin \text{Complement}(S_2[v_2]), \\ \alpha_1 & \text{if } S_1[u_1] = S_2[v_1] \text{ and } S_1[u_2] = S_2[v_2], \\ \alpha_2 & \text{if } S_1[u_1] = S_2[v_1] \text{ and } S_1[u_2] \neq S_2[v_2] \text{ or} \\ & \quad S_1[u_1] \neq S_2[v_1] \text{ and } S_1[u_2] = S_2[v_2], \\ \alpha_3 & \text{if } S_1[u_1] \neq S_2[v_1] \text{ and } S_1[u_2] \neq S_2[v_2]. \end{cases}$$

β, α_1, α_2 and α_3 are positive integer constants. Usually the parameters are set so that $\beta < \alpha_3 < \alpha_2 < \alpha_1$ which reflects that an arc alignment (α_1, α_2 or α_3) takes precedence over a single base alignment (β). Moreover, an arc alignment with exactly the same base pairs should score higher (α_1) since both the bases and their arcs are aligned. One can also have constraints on the arc width, for example, when $|u|$ or $|v|$ 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 as follows.

The common substructure of two arc-annotated sequences (S_1, P_1) and (S_2, P_2) is defined as the alignment between S_1 and S_2 where free positions in S_1 are aligned to free positions in S_2 and (both endpoints of) arcs in P_1 are aligned to (both endpoints of) arcs in P_2. The common substructure score is the weighted sum of all bases' and arcs' individual alignment scores. The Weighted Largest Common Substructure(WLCS) score is then defined as the maximum common substructure score among all possible common substructures. 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 [20]) 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). Recall that $|S_1| = n$ and $|S_2| = m$. Let $DP_{(i,i')}[j, j']$, where $1 \leq i \leq i' \leq n$ and $1 \leq j \leq j' \leq m$, denote the optimal WLCS score between $(S_1[i..i'], P_1)$ and $(S_2[j..j'], P_2)$ among all possible P_2. Note that $DP_{(i,i')}[j,j'] = 0$ whenever $i > i'$ or $j > j'$. Zhang presented an algorithm to compute $DP_{(1,n)}[1,m]$ that runs in $\mathrm{O}(nm^3)$ time and uses $\mathrm{O}(nm^2)$ space based on a two-step dynamic programming. Below are the three equations used to compute the two steps of the algorithm. Please refer to [20] for the correctness proofs.

Lemma 1. *(Lemma 4 in [20]) If i' is free,*

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

Lemma 2. *(Lemma 5 in [20]) For any arc $u \in P_1$ and $i < u_l$,*

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

Lemma 3. *(Lemma 3 in [20]) For any arc $u \in P_1$,*

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

Below we define three operations over the whole table $DP_{(i,i')}$, namely, EXTEND, MERGE, and ARC-MATCH.

Definition 1. *If i' is free 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($DP_{(i,i'-1)}$).*

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.*

Figure 1 describes the procedure WLCS(S_1, P_1, S_2) that computes $DP_{(1,n)}[j,j']$ for all $1 \le j \le j' \le m$. It is actually the algorithm in [20] expressed in terms of our defined operations on the DP tables. Given $DP_{(1,n)}[j,j']$ and all its intermediary DP tables, an optimal alignment can then be retrieved via the standard traceback procedure.

The time and space complexity of WLCS(S_1, P_1, S_2) is analyzed by computing the contributions of the operations EXTEND, ARC-MATCH, and MERGE separately. First we analyze the time complexity of the algorithm. An EXTEND operation involves computing $DP_{(i,i')}[j,j']$ from $DP_{(i,i'-1)}[j,j']$ for all $1 \le j \le j' \le m$. Since there are $O(m^2)$ (j,j') pairs to compute, each EXTEND operation takes $O(m^2)$ time. Next, because EXTEND is applied only on free positions, whose number is bounded by $O(n)$, the total cost for all EXTEND operations is $O(nm^2)$. The analysis for the ARC-MATCH operation is similar to the one for EXTEND above except that ARC-MATCH is invoked only on arcs whose cardinality is also bounded by $O(n)$ (since we assumed nested arc-annotation). Thus, it also takes $O(nm^2)$ time for all ARC-MATCH calls. Each call to MERGE requires computing the maximum $DP_{(i,i')}[j,j']$ by summing the values $DP_{(i,i'')}[j,j'']$ and $DP_{(i''+1,i')}[j''+1,j']$ where i'' is fixed and j'' is chosen from the range $[j..j']$.

WLCS(S_1, P_1, S_2)

For every arc $u \in P_1$ from the leftmost to the rightmost,

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($DP_{(u_l+1,u_r-1)}$).

Figure 1: The algorithm from [20] described in terms of EXTEND, MERGE and ARC-MATCH operations

In the worst case, one would require $O(m)$ time to compute $DP_{(i,i')}[j, j']$ for a particular (j, j'). This yields $O(m^3)$ time for a MERGE operation. Observing that the algorithm only invokes MERGE on arcs, the total contribution of MERGE is $O(nm^3)$. In total, the running time of the algorithm is $O(nm^3)$.

It is straightforward to see that EXTEND($DP_{(i,i'-1)}$) requires $O(m^2)$ space as we only need $O(m^2)$ space to store both $DP_{(i,i'-1)}$ and the resulting $DP_{(i,i')}$. The same argument also applies to ARC-MATCH and MERGE (as for MERGE, we need space for three DP tables instead of two). But since [20] uses the standard traceback for inferring the secondary structure of the sequence S_2, one must store all intermediary DP tables computed by WLCS(S_1, P_1, S_2). The size of the latter is bounded by $O(nm^2)$ as the number of free positions and arcs are both bounded by $O(n)$ and each DP table contains $O(m^2)$ entries.

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 above observation motivates the following definition.

Definition 4. *[6] For every row j of $DP_{(i,i')}$, a position j^* satisfying $j \leq j^* \leq m$ is called a* row interval point *if $DP_{(i,i')}[j, j^* - 1] < DP_{(i,i')}[j, j^*]$. (See Figure 2)*

Definition 5. *The set of row interval points j^* in the j^{th} row of $DP_{(i,i')}$ that satisfy $j^* \leq j'$ is denoted by* $\mathrm{RowIP}_{(i,i;j,j')}$.

Lemma 4. *For every $p \in [j..j']$, there exists a $j^* \in \mathrm{RowIP}_{(i,i';j,j')}$ such that $DP_{(i,i')}[j, j^*] = DP_{(i,i')}[j, p]$ and $j^* \leq p$.*

Proof. We know that the entries in any row of $DP_{(i,i')}$ are monotonically increasing. Hence each new distinct entry will be greater than the entry preceding it. By its definition, we can see that $RowIP_{(i,i';j,j')}$ covers all distinct entries in the interval $[j..j']$. $\square$

Lemma 5. *Let* $\alpha = \max\{\beta, \alpha_1, \alpha_2, \alpha_3\}$. *Then* $|\text{RowIP}_{(i,i';j,j')}| \leq \min\{\alpha(i'-i+1), (j'-j+1)\}$.

Proof. Since the row interval points are distinct, $|RowIP_{(i,i';j,j')}|$ is clearly bounded above by $j'-j+1$. Moreover, as we assume integer scores, the number of distinct interval points is also bounded above by the highest score possible from aligning $S_1[i..i']$ with $S_2[j..j']$, which is equal to $\min\{\alpha(i'-i+1), \alpha(j'-j+1)\}$. By combining the terms, the lemma follows. $\square$

In [20], for every (j,j') pair where $j \leq j'$, the procedure MERGE($DP_{(i,u_l-1)}, DP_{(u_l,u_r)}$) tries every possible $j'' \in [(j-1)..j']$ to compute the one that maximizes the sum

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

Given Lemma 5, we can see that there are at most $(\min\{\alpha(i'-i+1), (m-j+1)\})$ row interval points in any row j of $DP_{(i,i')}$. The following lemma implies that it is unnecessary to consider all $j'' \in [(j-1)..j']$ to find the maximum of (1).

Lemma 6. *The equation from Lemma 2 can be computed using the following equation*

$$DP_{(i,u_r)}[j,j'] = \max_{j^* \in \{\text{RowIP}_{(i,u_l-1;j,j')} \cup \{j-1\}\}} \{DP_{(i,u_l-1)}[j,j^*] + DP_{(u_l,u_r)}[j^*+1,j']\}.$$

Proof. Let us separate the range $[(j-1)..j']$ into $[(j-1)..(j-1)]$ and $[j..j']$. The lemma can be proven if we can show that, for every $j'' \in [j..j']$, there exists a $j^* \in RowIP_{(i,u_l-1;j,j')}$ such that $DP_{(i,u_l-1)}[j,j''] + DP_{(u_l,u_r)}[j''+1,j'] \leq DP_{(i,u_l-1)}[j,j^*] + DP_{(u_l,u_r)}[j^*+1,j']$. Note that, by Lemma 4, for each $j'' \in [j..j']$, there exists a $j^* \in RowIP_{(i,u_l-1;j,j')}$ such that $DP_{(i,u_l-1)}[j,j^*] = DP_{(i,u_l-1)}[j,j'']$ and $j^* \leq j'' \leq j'$. It follows that,

$$\begin{aligned} DP_{(i,u_l-1)}[j,j''] + DP_{(u_l,u_r)}[j''+1,j'] &= DP_{(i,u_l-1)}[j,j^*] + DP_{(u_l,u_r)}[j''+1,j'] \\ &\leq DP_{(i,u_l-1)}[j,j^*] + DP_{(u_l,u_r)}[j^*+1,j'] \end{aligned}$$

since by Observation 1(2), we know that $DP_{(u_l,u_r)}[j^*+1,j'] \geq DP_{(u_l,u_r)}[j''+1,j']$. $\square$

It is straightforward to see that the set $RowIP_{(i,u_l-1;j,j')}$ can be computed in $O(j'-j)$ time from the j^{th} row of $DP_{(i,u_l-1)}$. Hence before filling the entries of the new table $DP_{(i,u_r)}$, we precompute the sets $RowIP_{(i,u_l-1;j,m)}$ for $1 \leq j \leq m$ from the table $DP_{(i,u_l-1)}$, incurring an $O(m^2)$ time and space overhead. Lemma 6 speeds up the computation time of $DP_{(i,u_r)}[j,j']$ since we only consider distinct values for the $DP_{(i,u_l-1)}[j,j^*]$ terms. We further improve the time complexity of MERGE by also considering only distinct values of $DP_{(u_l,u_r)}[j^*+1,j']$. Let us start with the following definitions.

Definition 6. *Let us define the set*

$$\begin{aligned} S_{(i,i',i'';j,j')} &= \Big\{ j^* \in \text{RowIP}_{(i,i';j,j')} \cup \{j-1\} \mid j' \in \text{RowIP}_{(i'+1,i'';j^*+1,j')} \cup \{j^*\} \Big\}, \\ S'_{(i,i',i'';j,j')} &= \Big\{ \text{RowIP}_{(i,i';j,j')} \cup \{j-1\} \Big\} - S_{(i,i',i'';j,j')}, \end{aligned}$$

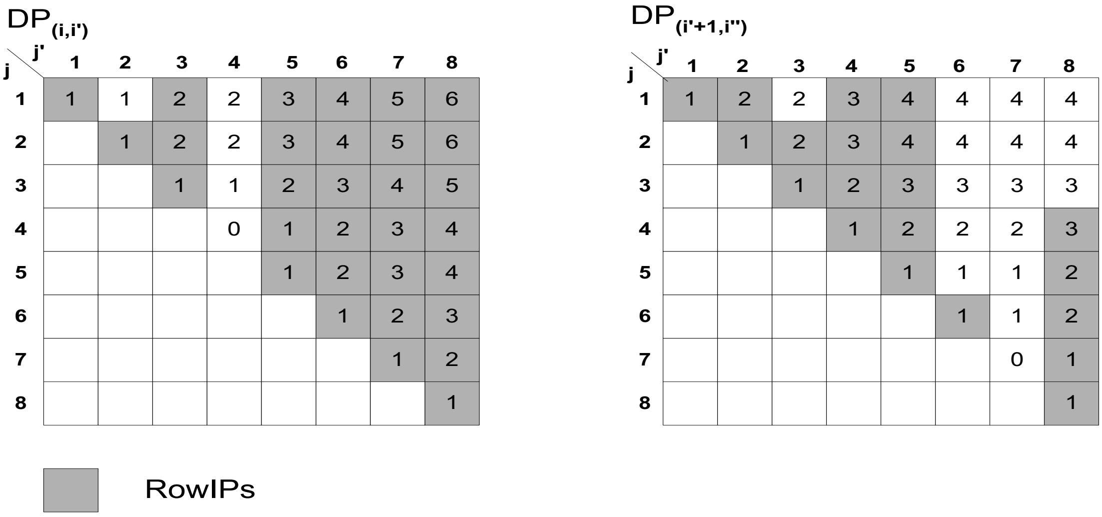

Figure 2: Illustration of the set S. From the figure we can see that $RowIP_{(i,i';2,8)} = \{2,3,5,6,7,8\}$ $(j=2,\ j'=8)$. Then, as defined, we have $S_{(i,i',i'';2,8)} = \{3,5,6,7,8\}$ since $j'=8$ and, $\forall x \in \{3,5,6,7\}$, $8 \in RowIP_{(i'+1,i'';x+1,8)}$.

Based on the set S and S' above, we define the following tables,

$$P_{(i,i',i'')}[j,j'] \quad = \quad \begin{cases} \max\limits_{j^* \in S_{(i,i',i'';j,j')}} \{DP_{(i,i')}[j,j^*] + DP_{(i'+1,i'')}[j^*+1,j']\} & \textit{if } S_{(i,i',i'';j,j')} \neq \emptyset, \\ 0 & \textit{otherwise} \end{cases}$$

$$P'_{(i,i',i'')}[j,j'] \quad = \quad \begin{cases} \max\limits_{j^* \in S'_{(i,i',i'';j,j')}} \{DP_{(i,i')}[j,j^*] + DP_{(i'+1,i'')}[j^*+1,j']\} & \textit{if } S'_{(i,i',i'';j,j')} \neq \emptyset, \\ 0 & \textit{otherwise} \end{cases}$$

The set $S_{(i,i',i'';j,j')}$ is actually a subset of the set $RowIP_{(i,i';j,j')} \cup \{j-1\}$ where for each of its element j^*, j' is in the set $RowIP_{(i'+1,i'';j^*+1,j')} \cup \{j^*\}$. Figure 2 illustrates the definition of the set $S_{(i,i',i'';j,j')}$. Given Definition 6 above, we can rewrite the equation in Lemma 6 into $DP_{(i,u_r)}[j,j'] = \max\{P_{(i,u_l-1,u_r)}[j,j'], P'_{(i,u_l-1,u_r)}[j,j']\}$. In the following lemma, we claim that we only need to compute the value of $DP_{(i,i')}[j,j^*] + DP_{(i'+1,i'')}[j^*+1,j']$ over $j^* \in S_{(i,i',i'';j,j')}$ instead of the whole $RowIP_{(i,i';j,j')}$ for each $j' \in [j..m]$.

Lemma 7. *When $P_{(i,u_l-1,u_r)}[j,j'] \leq P'_{(i,u_l-1,u_r)}[j,j']$, we have $DP_{(i,u_r)}[j,j'] = DP_{(i,u_r)}[j,j'-1]$.*

Proof. Given $P_{(i,u_l-1,u_r)}[j,j'] \leq P'_{(i,u_l-1,u_r)}[j,j']$, we have $DP_{(i,u_r)}[j,j'] = P'_{(i,u_l-1,u_r)}[j,j']$. To prove this lemma we shall show that $DP_{(i,u_r)}[j,j'-1] \leq P'_{(i,u_l-1,u_r)}[j,j']$ and $P'_{(i,u_l-1,u_r)}[j,j'] \leq DP_{(i,u_r)}[j,j'-1]$. The first one is trivial since, by Observation 1(1), $DP_{(i,u_r)}\ [j,j'-1] \leq DP_{(i,u_r)}[j,j'] = P'_{(i,u_l-1,u_r)}[j,j']$. Then we need to show that $P'_{(i,u_l-1,u_r)}[j,j'] \leq DP_{(i,u_r)}[j,j'-1]$. By its definition, $\forall j^* \in S'_{(i,u_l-1,u_r;j,j')}$, we have $j^* < j'$ and $j' \notin RowIP_{(u_l,u_r;j^*+1,j')}$. It follows that $\forall j^* \in S'_{(i,u_l-1,u_r;j,j')}$, we have

$$\begin{aligned} DP_{(u_l,u_r)}[j^*+1,j'] &= DP_{(u_l,u_r)}[j^*+1,j'-1], \text{ hence} \\ DP_{(i,u_l-1)}[j,j^*] + DP_{(u_l,u_r)}[j^*+1,j'] &= DP_{(i,u_l-1)}[j,j^*] + DP_{(u_l,u_r)}[j^*+1,j'-1]. \end{aligned}$$

```
MERGE(DP_(i,u_l-1), DP_(u_l,u_r))
Set P_(i,u_l-1,u_r)[j,j'] = 0 for 1 ≤ j ≤ j' ≤ m
for j = 1 ··· m
    for j* ∈ RowIP_(i,u_l-1;j,m) ∪ {j-1}
        for j' ∈ RowIP_(u_l,u_r;j*+1,m) ∪ {j*}
            P_(i,u_l-1,u_r)[j,j'] = max{P_(i,u_l-1,u_r)[j,j'], DP_(i,u_l-1)[j,j*]+DP_(u_l,u_r)[j*+1,j']}
        endfor
    endfor
    for j' = j ··· m
        DP_(i,u_r)[j,j'] = max{P_(i,u_l-1,u_r)[j,j'],  DP_(i,u_r)[j,j'-1]}
    endfor
endfor
```

Figure 3: The pseudocode for the new MERGE operation

Since $\mathit{RowIP}_{(i,u_l-1;j,j'-1)} = \mathit{RowIP}_{(i,u_l-1;j,j')} - \{j'\}$, $S'_{(i,u_l-1,u_r;j,j')} \subseteq \mathit{RowIP}_{(i,u_l-1;j,j')}$ and $j' \notin S'_{(i,u_l-1,u_r;j,j')}$, we have $S'_{(i,u_l-1,u_r;j,j')} \subseteq \mathit{RowIP}_{(i,u_l-1;j,j'-1)}$. Hence $P'_{(i,u_l-1,u_r)}[j,j'] \leq DP_{(i,u_r)}[j,j'-1]$. The lemma follows trivially. $\square$

Corollary 1. *We can compute the value of $DP_{(i,u_r)}[j,j']$ in Lemma 2 using the following equation*

$$DP_{(i,u_r)}[j,j'] = \max\{P_{(i,u_l-1,u_r)}[j,j'], DP_{(i,u_r)}[j,j'-1]\}.$$

The following lemma analyzes the complexity of the new MERGE operation.

Lemma 8. *The complexity of the new MERGE operation is in $O(\min\{\alpha(u_l-i),m\}\cdot\min\{\alpha|u|,m\}\cdot m)+O(m^2)$ time and $O(m^2)$ space.*

Proof. By Corollary 1, we can compute $DP_{(i,u_r)}[j,j']$ in constant time given that we have already computed the value of $P_{(i,u_l-1,u_r)}[j,j']$. A straightforward way to compute $P_{(i,u_l-1,u_r)}[j,j']$ is, for a particular j', compute the set $S_{(i,u_l-1,u_r;j,j')}$ and use it to compute the former based on Definition 6. This would take $O(\min\{\alpha(u_l-i),(j'-j)\}.\min\{\alpha(u_r-u_l),(j'-j)\})$ time. Taking all possible j and j', the running time will be in $O(\min\{\alpha(u_l-i),(j'-j)\}.\min\{\alpha(u_r-u_l),(j'-j)\}m^2)$, which is unacceptable.

To avoid the need of computing $S_{(i,u_l-1,u_r;j,j')}$, we reverse the computational ordering of j^* and j'. Instead of computing the values of j^* for each j'; for each $j^* \in \mathit{RowIP}_{(i,u_l-1;j,m)} \cup \{j-1\}$, we get the $j' \in \mathit{RowIP}_{(u_l,u_r;j^*+1,m)} \cup \{j^*\}$ and, for all such j', update the value of $P_{(i,u_l-1,u_r)}[j,j']$ whenever $DP_{(i,u_l-1)}[j,j^*]+DP_{(u_l,u_r)}[j^*+1,j'] > P_{(i,u_l-1,u_r)}[j,j']$. Effectively, for each $j' \in \mathit{RowIP}_{(u_l,u_r;j^*+1,j')}$ for some $j^* \in \mathit{RowIP}_{(i,u_l-1;j,j')}$, the updating will compute the maximum value of $DP_{(i,u_l-1)}[j,j^*]+DP_{(u_l,u_r)}[j^*+1,j']$ over all possible j^*. Note that we have to initialize the values in the table $P_{(i,u_l-1,u_r)}$ to zero beforehand.

The number of such (j^*,j') pair is bounded by $|\mathit{RowIP}_{(i,u_l-1;j,j')}|\cdot|\mathit{RowIP}_{(u_l,u_r;j^*+1,m)}|$ which is less than $\min\{\alpha(u_l-i),m\}\cdot\min\{\alpha|u|,m\}$. For each (j^*,j') pair, the sum $DP_{(i,i')}[j,j^*]+DP_{(i'+1,i'')}[j^*+1,j']$ will only be computed once taking constant time. As there are m rows in $P_{(i,u_l-1,u_r)}$, its time complexity will then be in $O(\min\{\alpha(u_l-i),m\}\cdot\min\{\alpha|u|,m\}\cdot m)$.

The size of $P_{(i,u_l-1,u_r)}$ is clearly in $O(m^2)$. Once we have computed $P_{(i,u_l-1,u_r)}$, we can compute the whole table of $DP_{(i,u_r)}$ in $O(m^2)$ time and space. By combining the complexity of the computation of both $P_{(i,u_l-1,u_r)}$ and $DP_{(i,u_r)}$, the lemma follows. $\square$

A MERGE operation can then be computed using the pseudocode in Figure 3.

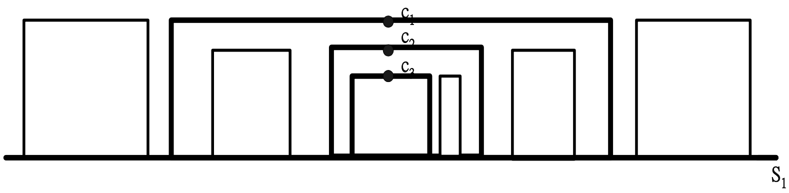

Figure 4: The core-path $CP(c_1)$ is the ordered set $\{c_1, c_2, c_3\}$

3.2 Complexity Analysis of the Improved MERGE Operation

As the sparsification technique only optimized the MERGE operations, the computational resources required by all EXTEND and ARC-MATCH operations remain the same as in Zhang's algorithm (Figure 1), i.e., $O(nm^2)$ for both time and space.

The previous section shows that each of the new MERGE($DP_{(i,u_l-1)}, DP_{(u_l,u_r)}$) operations requires $O(\min\{\alpha(u_l-i), m\}\cdot\min\{\alpha|u|, m\}\cdot m)+O(m^2)$ time and $O(m^2)$ space. We now consider the total time complexity of all MERGE operations. Let us start with some definitions to assist the analysis. The following is with respect to a *nested* arc-annotated structure.

Definition 7. *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 to as the* child *of the arc u. The set of children of an arc u is denoted by $Children(u)$.*

Definition 8. *A* terminal-arc *is defined to be an arc that has no child. A* core-arc*, with respect to an arc u, is a child of u that has the biggest size (arbitrarily breaking ties). The latter is denoted as core-arc(u). All other children of u are named the* side-arcs *and form the set* side-arcs(u).

Definition 9. *For any arc $u \in P_1$, the* core-path *$CP(u)$ is an ordered set of core-arcs $\{c_1, c_2, \ldots, c_\ell\}$, where $c_1 = u$ and for any c_i, c_{i+1} is core-arc(c_i) (refer to Figure 4).*

Lemma 9. *For any arc $u \in P_1$, the time required by the MERGE operations on all of its children in $Children(u)$ is in $\min\{O(\alpha(|u|-|c|)x_u m)+O(|Children(u)|m^2), O(|Children(u)|m^3)\}$ where c is the core-arc of u and $x_u = \min\{\alpha|u|, m\}$.*

Proof. The first observation is that MERGE only takes place when we encounter an arc as we try to extend the current DP table. Thus, the time required for applying MERGE on all arcs in $Children(u)$ is (by Lemma 8)

$$\sum_{u' \in Children(u)} \Big\{O(\min\{\alpha(u'_l - u_l), m\} \cdot \min\{\alpha|u'|, m\} \cdot m) + O(m^2)\Big\}.$$

The sum of the second term, $O(m^2)$, yields $O(|Children(u)|m^2)$ while the sum of the first term ($O(\min\{\alpha(u'_l - u_l), m\} \cdot \min\{\alpha|u'|, m\} \cdot m)$) gives several possible cases. When both $\min\{\alpha(u'_l - u_l), m\} = m$ and $\min\{\alpha|u'|, m\} = m$, the first term is equal to $O(m^3)$. Summing over all children of u gives $O(|Children(u)|m^3)$.

Otherwise, let $x_u = \min\{\alpha|u|, m\}$. We need to show that the summation of the first term is equal to $O(\alpha(|u| - |c|)x_u m)$. It is easy to show that $O(\min\{\alpha(u'_l - u_l), m\} \cdot \min\{\alpha|u'|, m\} \cdot m)$ is bounded above by $O(\alpha|u'|x_u m)$. Summing the value over $Children(u)$ only gives the bound of $O(\alpha|u|x_u m)$. To have a tighter bound, we separately consider the following cases;

1. The case when $|c| \le \frac{|u|}{2}$. For this case, $|u| - |c| > \frac{|u|}{2}$. Hence, $2(|u| - |c|) > |u|$ and we have $O(\alpha|u|x_u m) = O(\alpha(|u| - |c|)x_u m)$.

2. When $|c| > \frac{|u|}{2}$, applying MERGE on $DP_{(u_l+1,c_l-1)}$ and $DP_{(c_l,c_r)}$ will take at most $O(\alpha(|u|-|c|)x_u m)$ time since $\min\{(c_l-u_l),m\} \le \min\{(|u|-|c|),m\}$ and $min\{|c|,m\} \le x_u$. The remaining MERGE operations on the side-arcs will require at most $O(\alpha(|u|-|c|)x_u m)$ time too since their total size is bounded by $|u|-|c|$. Hence, in this case, the total time required is also bounded by $O(\alpha(|u|-|c|)x_u m)$.

□

Lemma 10. *The time required by all MERGE operations during the execution of $WLCS(S_1, P_1, S_2)$ is in* $\min\{O(\alpha^2 n^2 m + nm^2),\ O(\alpha nm^2 \log n),\ O(nm^3)\}$.

Proof. For convenience of notation, let us include an imaginary arc $r = (0, n+1)$ into P_1. Since the string S_1 is indexed from 1 to n, $S_1[0]$ and $S_1[n+1]$ are undefined and hence r will never be matched to any position in S_2. Note that r is the outermost arc and $|r| = O(n)$. Next, we define the set $Arc(y)$, where $y \in P_1$, to be the set $\{u \in P_1 | y_l < u_l < u_r < y_r\}$, that is, the set of all arcs in P_1 whose span is within $[y_l..y_r]$. Finally, based on Lemma 9, the time complexity $T(y)$ of all MERGE operations during the computation of $WLCS(S_1[y_l..y_r], P_1, P_2)$ can be computed by

$$T(y) = \sum_{\substack{u \in Arc(y)\\ c=\text{core-arc}(u)}} \min\Big\{O(\alpha(|u|-|c|)x_u m) + O(|Children(u)|m^2), O(|Children(u)|m^3)\Big\}. \tag{2}$$

$$= \sum_{\substack{u \in CP(y)\\ s \in \text{side-arcs}(u)}} T(s) + \sum_{\substack{u \in CP(y)\\ c=\text{core-arc}(u)}} \min\Big\{O(\alpha(|u|-|c|)x_u m) + O(|Children(u)|m^2), O(|Children(u)|m^3)\Big\}. \tag{3}$$

where $x_u = \min\{\alpha|u|, m\}$. We can derive (3) from (2) using the fact that

$$Arc(y) = CP(y) \cup \left(\bigcup_{\substack{u \in CP(y)\\ s \in \text{side-arcs}(u)}} Arc(s) \right).$$

Next we need to examine the following possible values of $\min\Big\{O(\alpha(|u|-|c|)x_u m) + O(|Children(u)|m^2), O(|Children(u)|m^3)\Big\}$.

1. $\min\Big\{O(\alpha(|u|-|c|)x_u m) + O(|Children(u)|m^2), O(|Children(u)|m^3)\Big\} = O(\alpha(|u|-|c|)x_u m) + O(|Children(u)|m^2)$. For this case we have

$$T(y) = \sum_{\substack{u \in CP(y)\\ s \in \text{side-arcs}(u)}} T(s) + \sum_{\substack{u \in CP(y)\\ c=\text{core-arc}(u)}} O(\alpha(|u|-|c|)x_u m) + \sum_{u \in CP(y)} O(|Children(u)|m^2).$$

$$\le \sum_{\substack{u \in CP(y)\\ s \in \text{side-arcs}(u)}} T(s) + \sum_{\substack{u \in CP(y)\\ c=\text{core-arc}(u)}} O(\alpha(|u|-|c|)x_y m) + \sum_{u \in CP(y)} O(|Children(u)|m^2). \tag{4}$$

$$= \sum_{\substack{u \in CP(y)\\ s \in \text{side-arcs}(u)}} T(s) + O(\alpha|y|x_y m) + \sum_{u \in CP(y)} O(|Children(u)|m^2). \tag{5}$$

We derive (5) from (4) by summing the telescoping series $\sum_{\substack{u\in CP(y)\\ c=\text{core-arc}(u)}} O(\alpha(|u|-|c|)x_y m)$. Next, depending on x_y,

(a) $x_y = \alpha|y|$

$$T(y) = \sum_{\substack{u\in CP(y)\\ s\in\text{side-arcs}(u)}} T(s) + O(\alpha^2|y|^2 m) + \sum_{u\in CP(y)} O(|Children(u)|m^2).$$

Since $|s| \leq \frac{|y|}{2}$ and $\sum_s |s| < |y|$, the recurrence yields a decreasing geometric series that sums up to $O(\alpha^2|y|^2 m)$ time complexity.

(b) $x_y = m$

$$T(y) = \sum_{\substack{u\in CP(y)\\ s\in\text{side-arcs}(u)}} T(s) + O(\alpha|y|m^2) + \sum_{u\in CP(y)} O(|Children(u)|m^2).$$

As $|s| \leq \frac{|y|}{2}$, the depth of recursion tree for the recurrence above is at most $O(\log |y|)$. And since $\sum_s |s| < |y|$, each level in the recursion tree will require less than $O(\alpha|y|m^2)$ time. Thus, in total, the time complexity of this case is $O(\alpha|y|m^2 \log |y|)$

2. $\min\left\{O(\alpha(|u|-|c|)x_u m) + O(|Children(u)|m^2), O(|Children(u)|m^3)\right\} = O(|Children(u)|m^3)$. In this case,

$$T(y) = \sum_{\substack{u\in CP(y)\\ s\in\text{side-arcs}(u)}} T(s) + \sum_{u\in CP(y)} O(|Children(u)|m^3). \tag{6}$$

which, by inspection, yields $T(y) = O(|y|m^3)$.

By taking y to be the (imaginary) arc r and combining all cases above, we conclude that $T(r) = \min\{O(\alpha^2 n^2 m + nm^2), O(\alpha nm^2 \log n), O(nm^3)\}$. ☐

Lemma 11. *Using the new MERGE operation, WLCS(S_1, P_1, S_2) runs in* $\min\{O(\alpha^2 n^2 m + nm^2), O(\alpha nm^2 \log n), O(nm^3)\}$ *time and* $O(nm^2)$ *space.*

Proof. As explained earlier, the operations EXTEND and ARC-MATCH both require $O(nm^2)$ time while the time complexity of MERGE is $\min\{O(\alpha^2 n^2 m + nm^2), O(\alpha nm^2 \log n), O(nm^3)\}$ by Lemma 10. Combining them will yield the time complexity stated in the lemma.

For the space complexity, assuming standard traceback, we have shown that EXTEND and ARC-MATCH operations will need $O(nm^2)$ space. A single MERGE operation will need $O(m^2)$ space as proven in Lemma 8. As MERGE is only applied on arcs, the total number of tables resulting from all MERGE operations is at most $O(n)$. The lemma thus follows. ☐

4 Improving the Space Complexity For Computing WLCS Score

In some cases, one is only interested to find the WLCS score. In this case, one would naturally expect a more space-efficient version of the WLCS routine as it is unnecessary to store old DP tables for traceback. Let us name such procedure as *the score-only* WLCS(S_1, P_1, S_2). It turns out that, using the original algorithm of Zhang [20], the space complexity is still bounded by $\Omega(nm^2)$ which is shown by the following lemma.

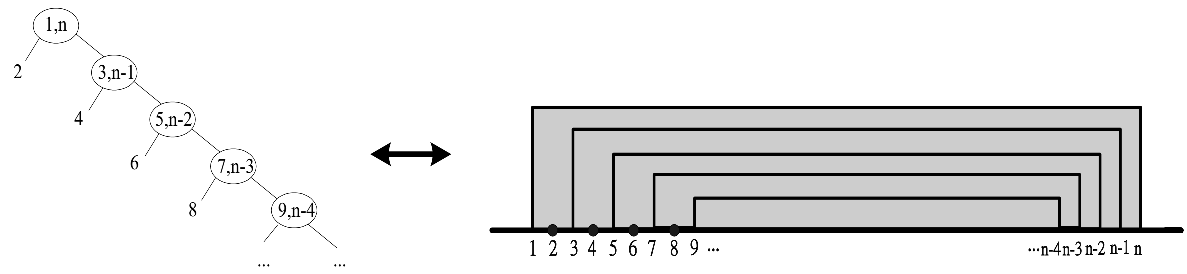

Figure 5: An example of arc-annotation on which the algorithm in [20] requires $\Omega(nm^2)$ space to compute the *score-only* WLCS(S_1, P_1, S_2). Note that the *post-ordering* forces the algorithm to compute the DPs for all the leaves before the internal nodes.

Lemma 12. *Using the original algorithm in [20] combined with the newly improved MERGE operation, the* score-only *WLCS(S_1, P_1, S_2) requires $\Omega(nm^2)$ space in the worst case.*

Proof. To compute the score-only WLCS(S_1, P_1, S_2), we only have to provide the space to perform the DP table operations, namely EXTEND, ARC-MATCH and MERGE and keep only the most current tables. As explained in Section 3, computing $DP_{(i,i')}$ =EXTEND$(DP_{(i,i'-1)})$ only requires $O(m^2)$ space provided that $DP_{(i,i'-1)}$ is already available when EXTEND is invoked. This condition is true for EXTEND and ARC-MATCH as we always compute $DP_{(i,i'-1)}$ before $DP_{(i,i')}$ and $DP_{(u_l+1,u_r-1)}$ before $DP_{(u_l,u_r)}$.

But this is not quite the same for MERGE operations. As described in [20], the routine WLCS(S_1, P_1, P_2) computes the DP tables according to the *post-order* of the nodes in the tree representing the sequence with the secondary structure. Given the post-order, whenever we execute MERGE $(DP_{(i,i'-1)}, DP_{(i',i'')})$, we would have computed $DP_{(i,i'-1)}$ but not $DP_{(i',i'')}$. While computing the latter, one must temporarily store $DP_{(i,i'-1)}$ in order to be able to finish the execution of the MERGE operation later. Note that the same kind of event could also take place during the computation of $DP_{(i',i'')}$. In the case of a skewed tree (see Figure 5), the number of temporarily stored DP tables can reach $\Omega(n)$(around $\frac{n}{3}$). Hence, $\Omega(nm^2)$ space is required. $\square$

4.1 Space Complexity Improvement by A Recursive Dynamic Programming Algorithm

This subsection will introduce a more-space efficient algorithm WLCS$_r(S_1, P_1, S_2)$ that computes the WLCS score using a carefully designed recursive dynamic programming algorithm. This improved algorithm guarantees that each MERGE operation is applied only to *side-arcs* where, by definition, the size of each side arc is at most half of the size of its parent.

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. As a special case, for the innermost core-arc $t \in CP(u)$ (which is a terminal arc), $DP_{(t_l,t_r)}$ can be computed without the MERGE operation. For the remaining core-arcs c, $DP_{(c_l,c_r)}$ will be computed using a *two-partition computation.* Let c' be core-arc(c) for an arc c. Due to the bottom-up ordering, $DP_{(c'_l,c'_r)}$ is computed before $DP_{(c_l,c_r)}$. 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 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$_r(S_1[s_l..s_r], P_1, S_2)$. Then we apply MERGE to combine $DP_{(s_l,s_r)}$ into the currently computed DP table. Having completed the computation of both phases, 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 ARC-MATCH($DP_{(c_l+1,c_r-1)}$).

If $(1,n) \in P_1$, then the largest arc u must be $(1,n)$ and we are done. Otherwise, we need to compute $DP_{(1,n)}$ using the same two-part computation technique: first compute $DP_{(1,u_l-1)}$, followed by $DP_{(u_l,n)}$, and then obtain $DP_{(1,n)}$ by MERGE($DP_{(1,u_l-1)}, DP_{(u_l,n)}$).

Lemma 13. *Computing WLCS$_r$(S_1, P_1, S_2) requires* $\min\{O(\alpha^2n^2m+nm^2), O(\alpha nm^2 \log n), O(nm^3)\}$ *time.*

Proof. As EXTEND and ARC-MATCH are still applied on free positions and arcs in S_1, respectively, the running time complexity of both operations are still the same as the one in Lemma 11 which are both in $O(nm^2)$.

Note that MERGE is now invoked on all arcs that 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. Lemma 8 has showed that MERGE($DP_{(i,u_l-1)}, DP_{(u_l,u_r)}$) takes $O(\min\{\alpha(u_l-i), m\} \cdot \min\{\alpha|u|, m\} \cdot m) + O(m^2)$ time to compute. Include an imaginary arc $r = (0, n+1)$ into P_1. Defining $T(u)$ ($u \in P_1$) as the total time complexity of MERGE during the computation of WLCS$_r$($S_1[u_l..u_r], P_1, P_2$), we can compute the total time complexity of all MERGE invocation in WLCS$_r$(S_1, P_1, S_2) by

$$
\begin{aligned}
T(r) \;&=\; \sum_{\substack{c\in CP(r)\\ s\in\text{side-arc}(c)}} \Big\{T(s) + O(\min\{\alpha(s_l-c_l), m\}\cdot\min\{\alpha|s|, m\}\cdot m) + O(m^2)\Big\}^* + \\
&\quad \sum_{\substack{c\in CP(r)\\ c'=\text{core-arc}(c)}} O(\min\{\alpha(c'_l-c_l), m\}\cdot\min\{\alpha(c_r-c'_l), m\}\cdot m) + O(m^2)^{\dagger}. \qquad (7)\\
&\le\; \sum_{\substack{c\in CP(r)\\ s\in\text{side-arc}(c)}} T(s) + \sum_{\substack{s\in\text{side-arc}(c)\\ c\in CP(r)}} O(\min\{\alpha n, m\}\cdot\min\{\alpha|s|, m\}\cdot m) + \\
&\quad \sum_{\substack{c\in CP(r)\\ c'=\text{core-arc}(c)}} O(\min\{\alpha(c'_l-c_l), m\}\cdot\min\{\alpha n, m\}\cdot m) + \\
&\quad \sum_{\substack{s\in\text{side-arc}(c)\\ c\in CP(r)}} O(m^2) + \sum_{c\in CP(r)} O(m^2). \qquad (8)\\
&=\; \min\{O(\alpha^2n^2m+nm^2), O(\alpha nm^2\log n), O(nm^3)\}. \qquad (9)
\end{aligned}
$$

To obtain (9) from (8), we make use the following facts

1. $\sum_{\substack{s\in\text{side-arc}(c)\\ c\in CP(r)}} |s| = \sum_{\substack{c\in CP(r)\\ c'=\text{core-arc}(c)}} (c'_l - c_l) = O(n)$ since, in all recursion level, all side-arcs $s \in$ side-arc(c), where $c \in CP(r)$, and the ranges $[c_l..c'_l]$ are non-overlapping. Hence, the sums of the term $\sum_{\substack{s\in\text{side-arc}(c)\\ c\in CP(r)}} O(\min\{\alpha n, m\}\cdot\min\{\alpha|s|, m\}\cdot m)$ and $\sum_{\substack{c\in CP(r)\\ c'=\text{core-arc}(c)}} O(\min\{\alpha(c'_l - c_l), m\}\cdot\min\{\alpha n, m\}\cdot m)$ in all recursion level would both be bounded by $\min\{O(\alpha^2n^2m)$, $O(\alpha nm^2\log n)$, $O(nm^3)\}$ (following a similar proof as in Lemma 10).

*Note that, here, MERGE is applied on $DP_{(c'_l,c_r-1)}$ while (c_l+1, c_r-1) is not an arc. Although this does not properly meet the definition of MERGE, the operation is still valid as MERGE basically combines any two DP tables without any specific requirement on the entries in any of the input tables.

*The time needed to compute $DP_{(s_l,s_r)}$ of side-arc s recursively and applying $MERGE(DP_{(i,s_l-1)}, DP_{(s_l,s_r)})$.

†The time needed to compute $MERGE(DP_{(c_l,c'_l-1)}, DP_{(c'_l,c_r)})$.

2. We can see that $\sum_{\substack{s \in \text{side-arc}(c) \\ c \in CP(r)}} m^2 + \sum_{c \in CP(r)} m^2 = \sum_{c \in CP(r)} |Children(c)| m^2$. Summing the term $\sum_{c \in CP(r)} |Children(c)| m^2$ over all recursion level will yield the bound of $O(nm^2)$.

□

Lemma 14. *$WLCS_r(S_1, P_1, S_2)$ uses $\min\{O(m^2 \log n), O(m^2 + \alpha mn)\} + O(n)$ space.*

Proof. Referring back to Lemma 12, we only need $O(m^2)$ to store the information needed to accomplish all EXTEND and ARC-MATCH operations. As for the MERGE operations, when there is no recursive call involved (the execution of MERGE on the LEFT and RIGHT parts), the space requirement is also in $O(m^2)$. In the recursive call, we now have managed to enforce a new computational ordering instead of using the original post-order (Lemma 12). Using the ordering given by the core-path in the annotation tree, Lemma 13 had shown that the latter guarantees $O(\log n)$ recursion level. Hence the number of temporarily stored $DP(i, s_l - 1)$ (s is a side-arc) during the recursive call to compute $DP_{(s_l, s_r)}$ will not exceed $O(\log n)$ as well. Storing only the row interval points takes $O(\min\{\alpha(s_l - i)m, m^2\})$ space (by Lemma 5) (with $O(m^2)$ time overhead for computing the set RowIP from/to the DP table). When $O(\min\{\alpha(s_l - i)m, m^2\}) = O(m^2)$, the space complexity is $O(m^2 \log n)$. For the other case, 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, $\sum_x O(\alpha(s_{l_x} - i_x)m) \le O(\alpha nm)$. Combining the two cases along with the space complexity of EXTEND, ARC-MATCH, we have $\min\{O(m^2 \log n), O(m^2 + \alpha mn)\}$. Finally, we add the space needed to store S_1,S_2 and P_1 which is in $O(n + m)$. The lemma follows. □

5 Hirschberg-like Traceback Algorithm

The previous section presents an algorithm $\text{WLCS}_r(S_1, P_1, S_2)$ to compute the WLCS score in $\min\{O(\alpha^2 n^2 m + nm^2), O(\alpha nm^2 \log n), O(nm^3)\}$ time and $\min\{O(m^2 \log n + n), O(m^2 + \alpha mn)\}$ space. Following the idea of Hirschberg in [11], this section presents an algorithm that computes the optimal WLCS alignment between (S_1, P_1) and (S_2, P_2) among all possible P_2 within the same time and space complexity. The outline of the algorithm is 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 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 so that, for any particular arc, both of its endpoints are in the same region. Given two points i_1 and i_2, $1 \le i_1 \le i_2 \le n$, 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 Figure 6). 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 represents 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, $0 < \phi < 1$.

Lemma 15. *Given a nested arc-annotated sequence S_1 of length n, we can compute two positions i_1 and i_2, $1 \le i_1 \le i_2 \le n$ in $O(n)$ time and space, such that i_1 and i_2 satisfy*

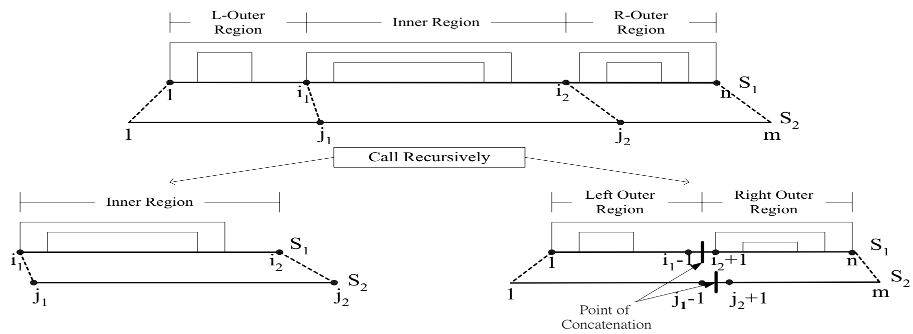

Figure 6: The recursion on the partitioned continuous region by Lemma 16.

1. $\frac{n}{3} \leq i_2 - i_1 + 1 \leq \frac{2n}{3}$,

2. i_1 *and* i_2 *are covered by the same arc* u*, or both are not covered by any arc,*

3. i_1 *is either a free position or the left endpoint of some arc* $u' \in Children(u)$

4. i_2 *is either a free position or the right endpoint of some arc* $u'' \in Children(u)$.

Proof. Define an imaginary arc $r = (0, n+1)$. Find a pair of core-arcs $c, c' \in CP(r)$ such that $c' = \text{core-arc}(c)$, $|c'| \leq \frac{2n}{3}$ and $|c| > \frac{2n}{3}$ (c could be r). When c is a terminal arc, i_1 and i_2 can be computed directly by choosing any two positions with distance at least $\frac{n}{3}$ and at most $\frac{2n}{3}$ in $[c_l..c_r]$.

Otherwise, if $\frac{n}{3} \leq |c'| \leq \frac{2n}{3}$, then we can use c'_l and c'_r as i_1 and i_2 (they are both covered by the core-arc c, i_1 is a left endpoint, and i_2 is a right endpoint). Else if $|c'| < \frac{n}{3}$, we first set i_1 and i_2 to c'_l and c'_r and increase the range $[i_1..i_2]$ by either increasing i_2 or decreasing i_1. Let us consider the case of increasing i_2. Suppose $i_2 + 1$ is a free position, then we can increase i_2 by 1. Else if $i_2 + 1$ is a left endpoint of some side-arc $s \in \text{side-arc}(c)$, then setting $i_2 = s_r$ will increase i_2 by $|s|$. Since $|s| < |c'| < \frac{n}{3}$, we guarantee that $|s| + |c'| < \frac{2n}{3}$.

Within this level of granularity, we can always extend the range $[i_1..i_2]$ until we have $\frac{n}{3} \leq i_2 - i_1 + 1 \leq \frac{2n}{3}$. At the same time, we will satisfy the remaining constraints since i_2 are chosen only from $C(c)$ and i_2 is never the left endpoint of any arc. The case of decreasing i_1 can be proven similarly. The time required by the steps above is at most $O(|CP(r)|) + O(|c|) = O(n)$ since finding c and c' takes $O(|CP(r)|)$, finding i_1 and i_2 takes $O(|c|)$ time and both $O(|CP(r)|)$ and $O(|c|)$ are at most in $O(n)$. All these operations can be performed in $O(n)$ space since we only need to store S_1 and P_1. □

Lemma 16. *We can always partition a continuous region into* 2 *non-overlapping subregions, where one of them is continuous and the other is gapped, in* $O(n)$ *time and space. Every subregion's size is at most* $\frac{2}{3}$ *of the original region.*

Proof. The proof of this lemma follows directly from Lemma 15. □

Definition 10. *Let the* ancestors *of an arc* u *be defined as the ordered set* $A(u) = \{u_1, u_2, u_3, ..u_\ell\}$ *where* $u_1 = u$ *and* $u_{i+1} = Parent(u_i)$*. Let the least common ancestor of the arcs* u *and* v*, denoted by* $LCA(u, v)$*, be the arc* $w \in A(u) \cap A(v)$ *where* $|w|$ *is minimal.*

Lemma 17. *We can always partition a gapped region into at most* 4 *non-overlapping subregions in* $O(n)$ *time and space. Every subregion's size is at most* $\frac{2}{3}$ *of the original region.*

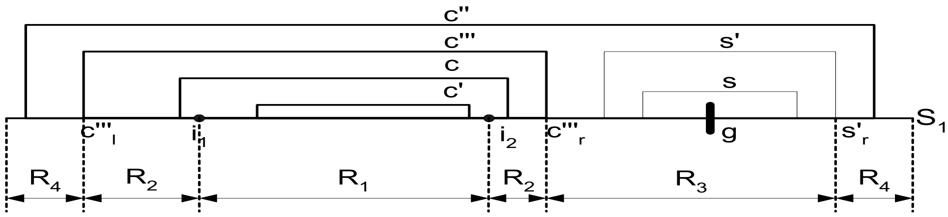

Figure 7: The figure describes the partitioning of S_1 for the case where $g > c_r$. For the sake of clarity, the regions are drawn connected to each other. Note that, actually, the regions R_1, R_2, R_3 and R_4 are disjoint (not sharing their endpoints).

Proof. Let $S_1[i_1..i_2]$ be a gapped region. First, as in Lemma 16, we compute the points i'_1 and i'_2 such that $\frac{(i_2-i_1+1)}{3} \leq i'_2 - i'_1 + 1 \leq \frac{2(i_2-i_1+1)}{3}$. Having computed such i'_1 and i'_2, we can guarantee that the size of $(i_2 - i_1 + 1) - (i'_2 - i'_1 + 1) \leq \frac{2(i_2-i_1+1)}{3}$. Let c and c' be the core-arcs where $c' =$ core-arc(c) and $c_l < i'_1 \leq c'_l < c'_r \leq i'_2 < c_r$. Further, let the position of the special gap character '$\star$' in $S_1[i_1..i_2]$ be denoted by g. Based several possible position of g with respect to i'_1, i'_2 and c; we have the following possible cases:

- $i'_1 \leq g \leq i'_2$. We have two *gapped* subproblems, $S_1[i'_1..i'_2]$ with g in it and $S_1[i_1..(i'_1-1)] \star S_1[(i'_2+1)..i_2]$.

- $c_l < g < i'_1$ or $i'_2 < g < c_r$. We will have one *continuous* region and two *gapped* regions. It is quite clear that the *continuous* region is $S_1[i'_1..i'_2]$. As for the gapped region, let us first consider the case where $c_l < g < i'_1$. If $g \in C(c)$, that is, g is a free position covered by c, then we have the *gapped* region $S_1[g..(i'_1-1)]$ and $S_1[i_1..(g-1)] \star S_1[(i'_2+1)..i_2]$. Else, if g is covered by some arc s, that is $g \in C(s)$, we find the ancestor of s that is a child of c. The latter is the arc s' such that $s' \in A(s) \cap Children(c)$. Then the first *gapped* region will be $S_1[s'_l..(i'_1-1)]$ and the second will be $S_1[i_1..(s'_l-1)] \star S_1[(i'_2+1)..i_2]$. The case where $i'_2 < g < c_r$ can be handled similarly.

- $g < c_l$ or $g > c_r$. In this case, we will have one *continuous* region, $S_1[i'_1..i'_2]$. In addition, we have three *gapped* regions. Suppose $g < c_l$. Let s be the arc that covers the position g. Let $c'' = LCA(s, c)$. It is clear that c'' is a core-arc too. Next, let $c''' =$ core-arc(c'') and s' be the arc in $A(s) \cap Children(c'')$. Now, we can readily define the gapped subproblems to compute in the next recursion. They are $S_1[c'''_l..(i'_1-1)] \star S_1[(i'_2+1)..c'''_r]$, $S_1[s'_l..(c'''_l-1)]$ and $S_1[i_1..(s'_l-1)] \star S_1[(c'''_r+1)..i_2]$. Again, the case where $g > c_r$ can be computed in the same fashion. Figure 7 illustrates the partitioning of S_1 in the case of $g > c_r$.

The running time of this case is still bounded by $O(n)$ since finding i'_1,i'_2, and the LCA of any two arcs requires at most $O(n)$ and they are executed in constant number of times. For the space requirement, again we will only need $O(n)$ space to store S_1 and P_1. □

From Lemmas 16 and 17, we can conclude that the computational complexity of the first step of our new algorithm is $O(n)$. 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. To do that, we will compute the positions in S_2 where the boundaries of each region are aligned to. We shall first show that we can compute such an alignment for one single position p in S_1. By definition, $DP_{(i,i')}[j, j']$ is the WLCS score produced by the optimal alignment between $S_1[i..i']$ and $S_2[j..j']$. Now, for each entry

$DP_{(i,i')}[j,j']$ in the table $DP_{(i,i')}$ where $i \leq p \leq i'$, we compute the position q, $j \leq q \leq j'$, such that either p is aligned to q or p is aligned to '$\sqcup$' and $[i..p-1]$ is aligned to $[j..q]$. We store such positions in a two dimensional table $A^p_{(i,i')}$ which is defined as follows,

Definition 11. *For $i \leq p \leq i'$ and $j \leq q \leq j'$, we define*

$$A^p_{(i,i')}[j,j'] = \begin{cases} q & \text{if } p \text{ is aligned to } q \text{ by } DP_{(i,i')}[j,j'], \\ -q & \text{if } p \text{ is aligned to } \sqcup \text{ and } [i..p-1] \text{ is aligned to } [i..q] \text{ by } DP_{(i,i')}[j,j'], \\ 0 & \text{if } DP_{(i,i')}[j,j'] \text{ does not align } [i..p] \text{ to any position in } S_2[j..j']. \end{cases}$$

During the computation of WLCS, the only time we will align a position p with some position q in S_2 is when we apply either $\chi(S_1[p], S_2[q])$ (when p is free), $\delta((S_1[p],..),(S_2[q],..))$, or $\delta((..,S_1[p]),(..,S_2[q]))$ (when p is an arc endpoint).

Lemma 18. *If p is free, then, for all $1 \leq j \leq j' \leq m$, we have*

$$A^p_{(i,p)}[j,j'] = \begin{cases} j' & DP_{(i,p)}[j,j'] = DP_{(i,p-1)}[j,j'-1] + \chi(S_1[p], S_2[j']), \\ -j' & DP_{(i,p)}[j,j'] = DP_{(i,p-1)}[j,j'] + \chi(S_1[p], \sqcup), \\ A^p_{(i,p)}[j,j'-1] & DP_{(i,p)}[j,j'] = DP_{(i,p)}[j,j'-1] + \chi(\sqcup, S_2[j']). \end{cases}$$

Proof. The first case in the recurrence is quite obvious since the optimal score $DP_{(i,p)}[j,j']$ is obtained by adding $DP_{(i,p-1)}[j,j'-1]$ with the score of aligning p with j' (by applying $\chi(S_1[p], S_2[j'])$). As for the second case, we know that p is aligned to $\sqcup$ and the alignment between $S_1[i..p]$ and $S_2[j..q]$ is actually the alignment corresponding to the score $DP_{(i,p-1)}[j,j']$. By Definition 11, we have $A^p_{(i,p)}[j,j'] = -j'$. Lastly, since the current j' is not included in the alignment, we must find the alignment of p in $S_2[j..j'-1]$. ☐

The case when p is not free (ARC-MATCH operation) can be handled similarly. Finally, for the case of MERGE operation and the case where $i < p < i'$, $A^p_{(i,i')}[j,j']$ is equal to $A^p_{(i'',i''')}[j',j'']$ where we have $i \leq i'' \leq p \leq i''' \leq i'$ and $DP_{(i,i')}[j,j'] = DP_{(i'',i''')}[j'',j'''] + X$ for X equals to some (probably empty) term that does not involve p (e.g the $\chi(S_1[i'], S_2[j'])$, $\delta((S_1[i], S_1[i']),(S_2[j], S_2[j'])$, or $DP_{(i'''+1,i')}[j'''+1,j'])$.

Lemma 19. *Given any position p, $1 \leq p \leq n$, we can compute the position q, $1 \leq q \leq m$, such that the optimal alignment between (S_1, P_1) and S_2 aligns either $S_1[1..p]$ to $S_2[1..q]$ or $S_1[1..p-1]$ to $S_2[1..q]$, within the same time and space complexity of the score-only $WLCS_r(S_1, P_1, S_2)$.*

Proof. Observe that the operation to compute the entry $A^p_{(i,i')}[j,j']$ can be done right after the computation of one particular $DP_{(i,i')}[j,j']$. Next, the recurrences above show that $A^p_{(i,i')}[j,j']$ can be computed in constant time. Hence computing $A^p_{(1,n)}[1,m]$ yields the same time complexity as computing $DP_{(1,n)}[1,m]$ which is the time complexity of WLCS$_r$.

As we only need to compute the value $A^p_{(1,n)}[1,m]$ for each position p, we do not have to store all of the intermediary tables $A^p_{(i,i')}$. Instead, as in the case of the score-only WLCS$_r(S_1, P_1, S_2)$, we only store those needed in the computation of the current $A^p_{(i,i')}[j,j']$. Consider the EXTEND operation. In computing $DP_{(i,p)} = \text{EXTEND}(DP_{(i,p-1)})$, we need to store $DP_{(i,p-1)}$. Correspondingly, computing $A^p_{(i,p)}[j,j']$ only requires the values in $A^p_{(i,p-1)}$. This also applies on the ARC-MATCH and MERGE operations.

Since, at any point of time, we only need the entries $A^p_{(i,i')}[j,j']$ from a constant number of (i,i') pairs (one pair for EXTEND and ARC-MATCH, two pairs for MERGE), we only need to store a constant number of such tables. Hence, the space needed by the $A^p_{(i,i')}$ table is also $O(m^2)$. ☐

Lemma 19 had shown that finding the alignment of a single point can be done within the same time and space complexity of the score-only $\text{WLCS}_r(S_1, P_1, S_2)$. Therefore, as the number of points to compute is at most a constant, the complexity of the second step of our algorithm is equal to the score-only $\text{WLCS}_r(S_1, P_1, S_2)$'s.

While applying the third step of our new algorithm (the recursive call) 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 $x \in \{A, C, G, U, \sqcup\}$.

Lemma 20. *Our new algorithm can recover the optimal WLCS alignment in* $\min\{O(\alpha^2 n^2 m + nm^2),\ O(\alpha n m^2 \log n),\ O(nm^3)\}$ *time and* $\min\{O(m^2 \log n)\ , O(m^2 + \alpha mn)\} + O(n)$ *space.*

Proof. Let $T(n, m)$ be the time complexity of the new algorithm. Let R_i denote the i^{th} region in S_1 on which the algorithm is recursively applied. Along with each region R_i, define R'_i to be the region in S_2 it is aligned to. We have earlier shown that the time complexity of the first and second step of our algorithm is in $\min\{O(\alpha^2 n^2 m + nm^2),\ O(\alpha n m^2 \log n),\ O(nm^3)\}$, hence we can formulate the recurrence

$$\begin{aligned} T(n, m) &= \sum_i T(|R_i|, |R'_i|) + \min\{O(\alpha^2 n^2 m + nm^2),\ O(\alpha n m^2 \log n),\ O(nm^3)\}. \\ &\leq \sum_i T(\frac{2}{3}n, |R'_i|) + \min\{O(\alpha^2 n^2 m + nm^2),\ O(\alpha n m^2 \log n),\ O(nm^3)\}. \qquad (10) \end{aligned}$$

where $\sum_i |R'_i| = m$. By inspection, we can see that the time complexity is still bounded by $\min\{O(\alpha^2 n^2 m + nm^2),\ O(\alpha n m^2 \log n),\ O(nm^3)\}$.

As for the space complexity, we define $S(n, m)$ to denote the space requirement of the algorithm. Each time after the second step of our algorithm, we must store the alignments computed in the latter. This requires a dedicated $O(n)$ space that can be accessed from all recursive calls. Observe that the space used by the current recursive level can be reused in the next level as we are only interested in the alignments of the regions that are being stored. Therefore,

$$\begin{aligned} S(n, m) &= \max\{\sum_i S(|R_i|, |R'_i|), \min\{O(m^2 \log n), O(m^2 + \alpha mn)\} + O(n)\}. \\ &\leq \max\{\sum_i S(\frac{2}{3}n, |R'_i|), \min\{O(m^2 \log n), O(m^2 + \alpha mn)\} + O(n)\}. \qquad (11) \end{aligned}$$

Again, by inspection, we show that the complexity of $S(n, m) = \min\{O(m^2 \log n),\ O(m^2 + \alpha mn)\} + O(n)$. The lemma thus follows. ☐

6 Concluding Remarks

Suppose we are given two homologous RNA sequences S_1 and S_2 where S_1 has a known structure. This paper studies the problem of inferring the structure of S_2 such that the WLCS score between the two structures is maximized. In the case of positive integer scoring, we designed an algorithm using dynamic programming sparsification technique that gives better time and space complexity than the brute-force approach.

Our techniques presented in this paper can be applied to *the longest arc-preserving common subsequence problem* (LAPCS) (see, e.g., [4, 5, 12]). Assuming similar scoring scheme (with the arc matching case removed, as the plain sequence would have no arc), we can also solve the LAPCS(nested, plain) problem in $\min\{O(nm^2+n^2m), O(nm^2\log n), O(nm^3)\}$ time and $\min\{O(m^2+mn), O(m^2\log n+n)\}$, thus improving the currently best known time and space complexity bounds for this problem ($O(nm^3)$ and $O(nm^2)$, respectively [12]).

One interesting extension of the problem discussed in this paper is to incorporate a more realistic, non-linear scoring function on the base and arc matching function. Another possible direction is to attack some special case 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. Computing the similarity of two sequences with nested arc annotations. *Theoretical Computer Science*, 312(2–3):337–358, 2004.

[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] P. A. Evans. Finding common subsequences with arcs and pseudoknots. In *CPM*, volume 1645, pages 270–280, 1999.

[6] W. Fu, W. K. Hon, and W. K. Sung. On all-substrings alignment problems. In *COCOON*, volume 2697, pages 80–89, 2003.

[7] J. Gramm, J. Guo, and R. Niedermeier. Pattern matching for arc-annotated sequences. In *FSTTCS*, volume 2556, pages 182–193, 2002.

[8] L. Grate. Automatic RNA secondary structure determination with stochastic context-free grammars. In *ISMB*, pages 136–144, 1995.

[9] 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.

[10] 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.

[11] D. S. Hirschberg. Algorithms for the longest common subsequence problem. *J. Association of Computing Machinery*, 24(4):664–675, 1977.

[12] T. Jiang, G. Lin, B. Ma, and K. Zhang. The longest common subsequence problem for arc-annotated sequences. *Journal of Discrete Algorithms*, 2(2):257–270, 2004.

[13] 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.

[14] 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.

[15] G. H. Lin, B. Ma, and K. Zhang. Edit distance between two RNA structures. In *RECOMB*, pages 211–220, 2001.

[16] R. B. Lyngsø, M. Zuker, and C.N.S. Pedersen. Internal loops in RNA secondary structure prediction. In *RECOMB*, pages 260–267, 1999.

[17] 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.

[18] 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.

[19] 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.

[20] K. Zhang. Computing similarity between RNA secondary structures. In *IEEE International Joint Symposia on Intelligence and Systems*, pages 126–132, 1998.

[21] M. Zuker. Prediction of RNA secondary structure by energy minimization. In *Methods in Molecular Biology*, volume 25, pages 267–94, 1994.

[22] 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.