2010

Improved Algorithms for Alignment between RNA Tertiary Structures

Qichan Ma
Western University

Improved Algorithms for Alignment between RNA Tertiary Structures

(Spine title: Algorithms for Alignment between RNA Tertiary Structures)

(Thesis format: Monograph)

by

Qichan Ma

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

CERTIFICATE OF EXAMINATION

THE UNIVERSITY OF WESTERN ONTARIO
School of Graduate and Postdoctoral Studies

The thesis by

Qichan Ma

entitled

Improved Algorithms for Alignment between RNA Tertiary Structures

is accepted in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

RNA is an important molecule which performs a wide range of functions in biological systems. The comparison between RNA secondary and tertiary structures has received much attention recently. It is a well known fact that structural features of RNAs are among the most significant factors in the molecular mechanisms involved in their functions. The presumption is that, to a preserved biological function there corresponds a preserved molecular structure. Therefore, the ability to compare RNA structures is useful. Furthermore, in many problems involving RNAs, it is required to have an alignment between RNA structures in addition to a similarity measure.

Computing alignment between RNA tertiary structures is NP-hard and MAX SNP-hard. In this research, we present algorithms for computing the alignment between two RNA tertiary structures. For simple tertiary structures, we can compute the optimal alignment efficiently. For moderate tertiary structures, we adopt the constrained alignment approach. Although the result produced by constrained alignment is not guaranteed to be an optimal solution, in practice it would be reasonable. Experimental tests show that our algorithms can be used to compute alignment between RNA tertiary structures in practical applications.

Key words: dynamic programming, sequence alignment, RNA, RNA secondary structure, RNA tertiary structure, RNA structural alignment.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor, Dr. Kaizhong Zhang, for his guidance, support and encouragement during my master study and research. I could not have imagined having a better advisor and mentor for my master study.

I sincerely thank the members of the examination board for their helpful suggestions.

I would like to thank my family for their love and support.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Objective and Motivation

In this research, the problem of interest is structural alignment between two RNA molecules. More specifically, we focus on alignment between two RNA tertiary structures.

The comparison and alignment between RNA secondary and tertiary structures has received much attention recently. The motivation for this kind of work is mainly due to the importance of RNA molecule. RNA is an important molecule which performs a wide range of functions in biological systems. At the ribosomal level, messenger RNA (mRNA) is used to read the genetic code and transfer RNA (tRNA) is utilized to make the protein sequence. In the case of certain viruses, such as HIV, it is RNA (not DNA) that carries genetic information and regulates the functions of these viruses. RNA has recently become the center of much attention due to its catalytic properties, leading to an increasing interest in obtaining structural information.

It is a well known fact that structural features of RNAs are among the most significant factors in the molecular mechanisms involved in their functions. The presumption is that, to a preserved biological function there corresponds a preserved molecular structure. Hence the ability to compare RNA structures is useful [1, 5, 6, 8, 11, 21].

Furthermore, in many problems involving RNAs [3, 16], it is required to have an alignment between RNA structures in addition to a similarity measure [17].

The primary structure of RNA is a sequence of nucleotides (bases). The primary structure, also called RNA sequence, is denoted by a sequence over the four-letter alphabet $\Sigma = \{A, C, G, U\}$. The secondary or tertiary structure of RNA is a set of base-pairs. For RNA secondary structures, these base-pairs have traditionally been assumed to be nested, i.e. one-to-one and non-crossing. For tertiary structure, there is no restriction of non-crossing.

Arc-annotated sequences [6, 8, 9, 13] are useful in representing the structural information of RNA sequences. In general, RNA secondary and tertiary structures can be represented as a set of nested arcs and a set of crossing arcs, respectively.

Computing alignment between RNA tertiary structures is NP-hard and MAX SNP-hard [21, 12, 19, 9]. This means that this problem has no solutions in polynomial time, and there is no polynomial time approximation scheme (PTAS) for it unless P = NP.

Wang and Zhang presented an algorithm to compute the alignment between RNA structures for the case where aligned base pairs are non-crossing [19]. They treated a base pair as a unit and do not allow it to match to two unpaired bases. Under this restricted assumption, when at least one of the RNA structures involved is a secondary structure, their algorithm can compute the optimal alignment in $O(\mathit{stem}(R_1) \times \mathit{stem}(R_2) \times |R_1| \times |R_2|)$ time. One more step, can be added to the algorithm to align tertiary base pairs. This step can be considered as a constrained alignment.

An improved edit distance model was proposed by Jiang *et al.* in [9] to measure the similarity between RNA structures. In addition to base insertion, base deletion, base substitution, base-pair insertion, base-pair deletion, base-pair substitution, the authors introduced two new edit operations, base-pair bond breaking and base-pair altering. Under this model, even computing edit distance between a tertiary structure and a primary structure is MAX SNP-hard. Jiang *et al.* present a dynamic programming algorithm for computing edit distance between a secondary structure and a primary structure, and a dynamic programming algorithm for a solvable case of edit distance between a tertiary structure and a secondary structure under a class of restricted scoring schemes [9].

Another line of works related to similarity comparison between RNA secondary and tertiary structures is focused on primary structure [1, 5] where the comparison is basically done on the primary structure while trying to incorporate the secondary and tertiary structural information. The weakness of this approach is that it does not treat a base-pair as a whole entity.

Recently, a fixed parameter tractable algorithm was proposed in [13]. This algorithm is a generalization of the algorithm in [9] to tertiary structures. It computes the optimal alignment between two RNA tertiary structures. The parameter, which determines the exponential runtime, depends on how complex the crossing stems are arranged. Unfortunately, this algorithm only works if the fixed parameter k is very small, for example $k = 1$. When the parameter is large, it is not affordable due to too high usage of space and time.

In this thesis, we present algorithms for computing the alignment between two RNA tertiary structures. We follow the work of Möhl *et al.* [13], and have made several optimizations to accelerate their algorithm. For simple tertiary structures, we can compute the optimal alignment efficiently. For moderate tertiary structures, we adopt the constrained alignment approach. Although the result produced by constrained alignment is not guaranteed to be an optimal solution, in practice it would be reasonable. Experimental tests show that our algorithm can be used to compute alignment between RNA tertiary structures in practical applications.

The rest of the thesis is organized as follows.

In the remaining part of this chapter, we provide a brief introduction to the RNA structure.

Chapter 2 gives an overview of sequence alignment. It serves as a useful background for RNA structural alignment problem which is discussed in Chapter 3.

Chapter 3 gives a review of RNA structural alignment. It serves as a useful background for our algorithms.

In Chapter 4, we present algorithms for alignment between two RNA tertiary structures. This work is based on [13] by Möhl *et al.*.

Chapter 5 presents the detail of implementation and a discussion of experimental results.

In Chapter 6, several conclusions are drawn and suggestions for future work are given.

1.2 RNA Structures

An RNA (Ribonucleic Acid) is a polymer consisting of ribonucleotides linked together in a chain. Each ribonucleotide contains one of four possible bases, which are adenine (A), cytosine (C), guanine (G) and uracil (U).

An RNA molecule has two sets of structural information. First, the *primary structure* of an RNA molecule is a single strand made of the ribonucleotides A, C, G and U. Second, the ribonucleotide sequence folds over onto itself principally by means of hydrogen bonds to form double-stranded regions of base pairings, yielding higher order *secondary structure* or *tertiary structure.*

Because an RNA sequence is composed of four possible bases, we can use a four-letter alphabet $\Sigma = \{A, C, G, U\}$ to represent an RNA sequence. This base sequence is usually referred to as *primary structure.* Formally, the character sequence $R = r_1 r_2 \cdots r_n$ where $r_i \in \Sigma$ $(1 \leq i \leq n)$ is called *RNA primary structure.* Conventionally, we will refer to the left end of the sequence as the $5'$ end of the RNA and the right end of the sequence as the $3'$ end of the RNA.

RNA secondary and tertiary structures are represented as a set of bonded pairs of bases. Various base pairings have been detected, but three kinds occur more frequently than any others. These three base pairings are $A - U$ $(U - A)$, $C - G$ $(G - C)$ and $G - U$ $(U - G)$. Base pairs $A - U$ $(U - A)$ and $C - G$ $(G - C)$ are called

Watson-Crick base pairs. The base pair $G - U$ ($U - G$) is referred to as *Wobble base pair.* These three types of pairings are referred to as *canonical base pairs.* Others are called *non-canonical base pairs.*

A bonded pair of bases (*base-pair*) is usually represented as an arc between the two complementary bases involved in the bond. It is assumed that any base participates in at most one such pair (i.e., one-to-one). If there is no crossing in the set of arcs representing a structure, the structure is considered as *secondary structure.* Otherwise, the structure is considered as *tertiary structure.* The crossing pairs form *pseudoknots.* An example of secondary structure is given in Figure 1.1. Tertiary structure and pseudoknots are illustrated in Figure 1.2.

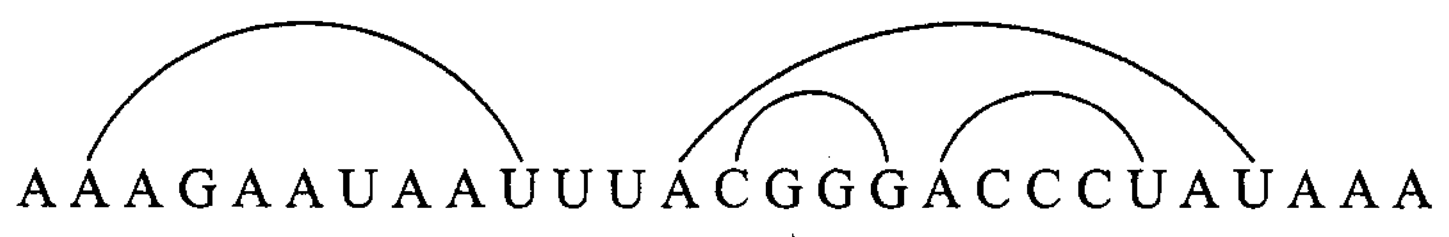


Figure 1.1: An example of RNA secondary structure

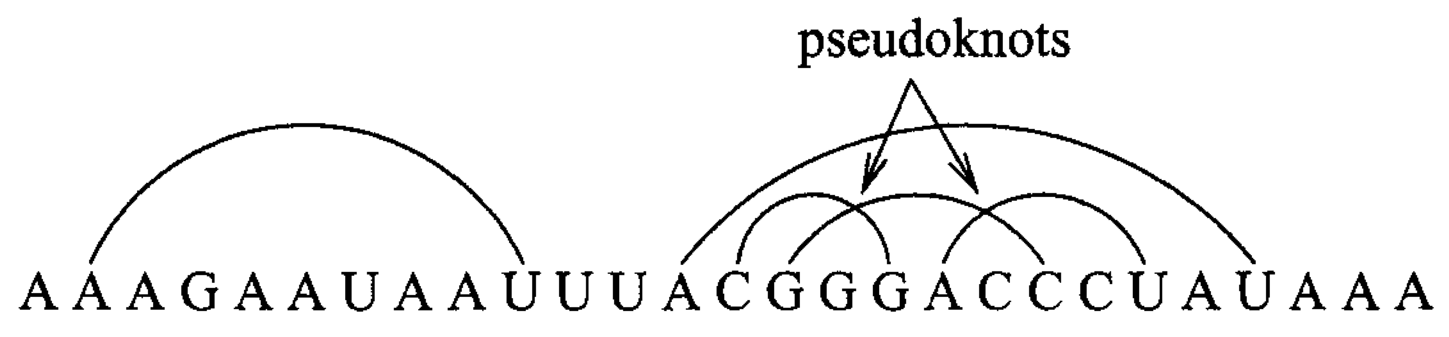


Figure 1.2: An example of RNA tertiary structure and pseudoknots

In an RNA structure, it occurs frequently that base pairs are stacked up one next to another. These stacked base pairs together form a *stem*, as shown in Figure 1.3. In the traditional definition, the stem is stacked pairs of maximal length. We will introduce a different definition in Chapter 3.

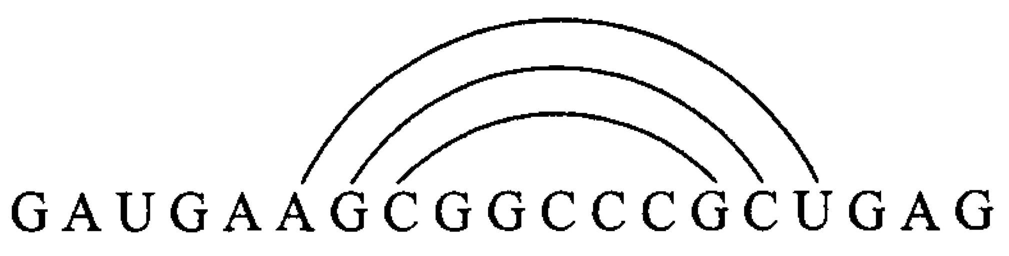


Figure 1.3: An example of stem

Chapter 2

A Review of Sequence Alignment

In this chapter, we give an overview of the most foundation problem of biological sequence analysis — sequence alignment. Most algorithms discussed in this chapter are extended from Zhang's course notes [20].

A sequence can be viewed as a primary structure without base pairs. Thus the structural alignment problem which we will discuss in Chapter 3 is actually an extension to the sequence alignment problem.

2.1 Alignment

In this thesis, we will not touch local alignment or fitting one sequence into another. We only discuss global alignment which serves as a basis for the structural alignment problem in Chapter 3. When there is no confusion, global alignment will be referred to as just "alignment".

A sequence alignment is a possible way in which characters of one sequence may be matched with characters of another sequence. It shows one way, out of many, to edit one sequence into the other. It can show how similar two sequences are.

Now we give a formal definition of alignment.

If S is a sequence, then $|S|$ denotes the length of S and $S[i]$ denotes the i-th character of S. We use $S[i, j]$ to represent the subsequence of S from $S[i]$ to $S[j]$. We use symbol $'-'$ to denote space.

Given two sequences S_1 and S_2 which are over some alphabet Σ, the alignment of S_1 and S_2 is represented by (S_1', S_2') satisfying the following conditions:

- S_1' is S_1 with some new symbol $'-'$ which is not in Σ inserted and S_2' is S_2 with some new symbol $'-'$ inserted such that $|S_1'| = |S_2'|$.
- $\forall i \in \{1, \ldots, |S_1'|\}$, at least one of $S_1'[i]$ and $S_2'[i]$ cannot be $'-'$.

In the case of RNA sequence alignment, $\Sigma = \{A, C, G, U\}$.

2.1.1 Edit Operations

The most popular measurements that are used to compare the similarity of two sequences are *distance* and *similarity*. In this research, we adopt the distance measure.

There are three basic edit operations, *insertion*, *deletion* and *substitution* in sequence alignment. They are defined as follows.

- Insertion: insert a character into a sequence.
- Deletion: delete a character from a sequence.
- Substitution: replace a character in a sequence with another character. There are two types of substitution. One is match (a character is replaced by an identical character), and the other is mismatch (a character is replaced by a different character).

Now we give a formal presentation. We represent an edit operation as $a \to b$, where a and b are either λ, the null label, or labels from the alphabet Σ.

We call $a \to b$ a substitute operation if $a \neq \lambda$ and $b \neq \lambda$ (if $a = b$, it is a match operation; else it is a mismatch operation); a delete operation if $b = \lambda$; and an insert operation if $a = \lambda$. (Figure 2.1 gives a simple illustration of sequence alignment with edit operations.) Let Γ be a cost function which assigns to each edit operation $a \to b$ a nonnegative real number $\Gamma(a \to b)$. We constrain Γ to be a distance metric. That is, (1) $\Gamma(a \to b) \geq 0$, $\Gamma(a \to a) = 0$, (2) $\Gamma(a \to b) = \Gamma(b \to a)$, and (3) $\Gamma(a \to c) \leq \Gamma(a \to b) + \Gamma(b \to c)$.

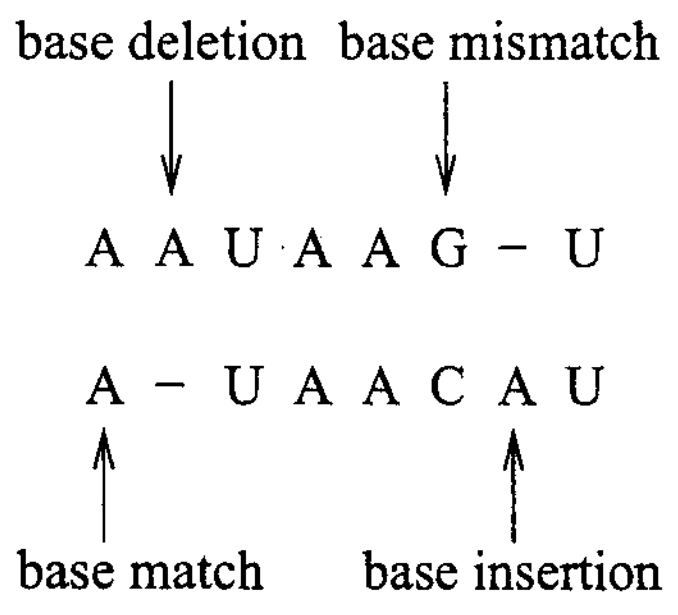


Figure 2.1: Sequence alignment with edit operations

Given an alignment (S'_1, S'_2), we define substitution M, deletion D, insertion I, as follows.

$$\begin{aligned}
M &= \{\ i \mid S'_1[i] \text{ and } S'_2[i] \text{ are characters from } \Sigma\}.\\
D &= \{\ i \mid S'_1[i] \text{ is a character from } \Sigma \text{ and } S'_2[i] = '-'\}.\\
I &= \{\ i \mid S'_2[i] \text{ is a character from } \Sigma \text{ and } S'_1[i] = '-'\}.
\end{aligned}$$

The cost of an alignment (S'_1, S'_2) is defined as follows.

$$\begin{aligned}
cost((S'_1, S'_2)) = &\sum_{i \in M} \Gamma(S'_1[i] \to S'_2[i])\\
&+ \sum_{i \in D} \Gamma(S'_1[i] \to \lambda)\\
&+ \sum_{i \in I} \Gamma(\lambda \to S'_2[i]) \qquad (2.1)
\end{aligned}$$

Given two sequences S_1 and S_2, the edit alignment between them is defined as

$$Align(S_1, S_2) = \min_{(S'_1, S'_2)} \{cost((S'_1, S'_2))\}. \qquad (2.2)$$

A similarity (maximization) version can also be considered, where the goal is to find the maximum-scoring edit alignment. It is similar to the distance alignment problem. We still define three edit operations: *insertion*, *deletion* and *substitution*, which are the same as the ones in the distance version. The major difference is the scoring method. In the similarity alignment problem, a mismatch is assigned a real score, a match is assigned a positive score, and each deletion/insertion is assigned a non-positive score. The similarity alignment problem was proposed and solved by Needleman and Wunsch in [14]. We will explain a variation of their algorithm under the distance measure later.

Finding distance and similarity alignments are dual problems. That is, when aligning two sequences by distance, there is a similarity algorithm that gives the same optimal alignment and vice versa. A score from distance alignment or similarity alignment can indicate the homology between sequences. Smaller distance score or higher similarity score reflects the higher degree of homology.

2.1.2 Number of Alignments

A naive way to find the optimal alignment between two sequences is to try all possible combinations of edit operations and pick the best one. However, the number of all possible alignments is exponential with respect to the sequence length. Consider two sequences $S = s_1 s_2 \ldots s_n$ and $T = t_1 t_2 \ldots t_m$. An alignment between these two sequences can end in only three ways:

$$\begin{array}{ccc} \cdots s_n & \cdots s_n & \cdots - \\ \cdots - & \cdots t_m & \cdots t_m \end{array}$$

Define $f(n,m)$ to the number of possible alignments between sequence S with length n and T with length m. Therefore we have the recursion

$$f(n,m) = f(n-1,m-1) + f(n-1,m) + f(n,m-1) \tag{2.3}$$

If s_n is aligned to t_m, there are $f(n-1, m-1)$ ways of aligning $s_1 s_2 \ldots s_{n-1}$ to $t_1 t_2 \ldots t_{m-1}$. If s_n is aligned to $'-'$, there are $f(n-1, m)$ ways of aligning $s_1 s_2 \ldots s_{n-1}$ to $t_1 t_2 \ldots t_m$. If t_m is aligned to $'-'$, there are $f(n, m-1)$ ways of aligning $s_1 s_2 \ldots s_n$ to $t_1 t_2 \ldots t_{m-1}$.. When $m = n$, $f(n,n)$ can be solved as

$$f(n,n) \approx (1+\sqrt{2})^{2n+1}\sqrt{n} \tag{2.4}$$

This approximation is obtained by H. T. Laquer [10].

For example, given two sequences each of length 500, there are

$$f(500,500) \approx (1+\sqrt{2})^{1001}\sqrt{500} \approx 3.22 \times 10^{384} \text{ possible alignments.}$$

This example shows that it is just not feasible to enumerate all possible alignments to get the optimal one. Therefore, a better algorithm is needed to compute optimal alignment between two sequences.

2.1.3 Dynamic Programming Solution

2.1.3.1 Dynamic Programming

Dynamic programming is an efficient programming technique for solving a broad range of search and optimization problems which exhibit the characteristics of *optimal substructure* and *overlapping subproblems* [4]. A problem exhibits *optimal substructure* if an optimal solution to the problem can be constructed from optimal solutions to its subproblems [4]. A problem has *overlapping subproblems* if it can be broken down into subproblems which are reused multiple times [4]. Dynamic programming has long been a major technique in the sequence alignment problem [14, 18].

The idea of dynamic programming is to solve a problem by first solving its subproblems. The smallest subproblems are explicitly solved first, and the results of these are used to construct solutions to progressively larger subproblems. This constitutes a bottom-up approach.

2.1.3.2 Property of Optimal Alignments

Consider two sequences S_1 and S_2. We can observe that a prefix of the optimal alignment between S_1 and S_2 which contains exactly first i characters of S_1 and first j characters of S_2 (this prefix may also contain spaces '$-$'s) is an optimal alignment between a prefix $S_1[1,i]$ of S_1 and a prefix $S_2[1,j]$ of S_2. So an optimal alignment score can be computed by scanning S_1 and S_2 from left to right, recording only the optimal alignment scores between prefixes of S_1 and S_2.

We use $\Gamma(\ )$ to define $\gamma(i,j)$ for $0 \le i \le |S_1|$ and $0 \le j \le |S_2|$.

$$\gamma(i,0) = \Gamma(S_1[i] \rightarrow \lambda) \tag{2.5}$$

$$\gamma(0,j) = \Gamma(\lambda \rightarrow S_2[j]) \tag{2.6}$$

$$\gamma(i,j) = \Gamma(S_1[i] \rightarrow S_2[j]) \tag{2.7}$$

From this definition, we know that $\gamma(i,0)$ is the cost of deleting character $S_1[i]$, $\gamma(0,j)$ is the cost of inserting character $S_2[j]$, and $\gamma(i,j)$ is the cost of aligning $S_1[i]$ to $S_2[j]$.

We now consider the optimal alignment between $S_1[1,i]$ and $S_2[1,j]$. We use $A(i,j)$ to represent the optimal alignment cost between $S_1[1,i]$ and $S_2[1,j]$. The following lemmas will show how to compute $A(i,j)$.

Lemma 2.1.1

$$A(0,0) = 0 \tag{2.8}$$

Proof: Consider $A(1,1)$. If the optimal alignment results from aligning $S_1[1]$ to $S_2[1]$, then we only need to account for the cost for aligning $S_1[1]$ to $S_2[1]$. Hence we may set $A(0,0) = 0$. $\square$

Lemma 2.1.2 *For* $i > 0$,

$$A(i,0) = A(i-1,0) + \gamma(i,0) \tag{2.9}$$

Proof: It is obvious that each element in $S_1[1,i]$ is aligned to '$-$'. That is, $S_1[i]$ is aligned to '$-$', and each element in $S_1[1,i-1]$ is aligned to '$-$'. Hence we have $A(i,0) = A(i-1,0) + \gamma(i,0)$. □

Lemma 2.1.3 *For $j > 0$,*

$$A(0,j) = A(0,j-1) + \gamma(0,j) \tag{2.10}$$

Proof: Similar to Lemma 2.1.2. □

Lemma 2.1.4 *For $i > 0$ and $j > 0$,*

$$A(i,j) = \min \begin{cases} A(i-1,j) + \gamma(i,0) \\ A(i,j-1) + \gamma(0,j) \\ A(i-1,j-1) + \gamma(i,j) \end{cases} \tag{2.11}$$

Proof: Consider $S_1[i]$ and $S_2[j]$. There are exactly the following cases.

(1) $S_1[i]$ is aligned to '$-$'. Thus $S_1[1,i-1]$ is aligned to $S_2[1,j]$. Hence the $A(i-1,j) + \gamma(i,0)$ item.

(2) $S_2[j]$ is aligned to '$-$'. Thus $S_1[1,i]$ is aligned to $S_2[1,j-1]$. Hence the $A(i,j-1) + \gamma(0,j)$ item.

(3) $S_1[i]$ is aligned to $S_2[j]$. Thus $S_1[1,i-1]$ is aligned to $S_2[1,j-1]$. Hence the $A(i-1,j-1) + \gamma(i,j)$ item.

Therefore we take the minimum of the three cases and get the above recursion. □

2.1.3.3 Algorithm

From Lemmas 2.1.1 to 2.1.4, we can compute $Align(S_1, S_2) = A(|S_1|, |S_2|)$ using a bottom-up approach. Algorithm 2.1 shows how to compute optimal alignment score.

The implementation involves filling a matrix $M(0..|S_1|; 0..|S_2|)$ of size $(|S_1| + 1)(|S_2| + 1)$. We use Eq. 2.8 to 2.11 to fill the matrix, starting at the upper-left

Algorithm 2.1 *Sequence-Distance*(S_1, S_2)

Input: Two sequences S_1 and S_2 with $n = |S_1|$ and $m = |S_2|$.
Output: Alignment score matrix $M(0..|S_1|; 0..|S_2|)$.

```
1: compute $A(0,0)$ as in Lemmas 2.1.1
2: for $i \leftarrow 1$ to $n$ do
3:   compute $A(i,0)$ as in Lemmas 2.1.2
4: end for
5: for $j \leftarrow 1$ to $m$ do
6:   compute $A(0,j)$ as in Lemmas 2.1.3
7: end for
8: for $i \leftarrow 1$ to $n$ do
9:   for $j \leftarrow 1$ to $m$ do
10:    compute $A(i,j)$ as in Lemmas 2.1.4
11:  end for
12: end for
```

cell and scan the matrix from left to right, row by row as we are filling it. A matrix entry is assigned a value based on its adjacent (top, left and top-left) entries of which the values have been computed.

The time complexity of Algorithm 2.1 is $O(|S_1||S_2|)$, and the space complexity is $O(|S_1||S_2|)$.

To produce an optimal alignment, we *trace back* the matrix containing alignment scores. We start at the matrix cell holding the optimal score (that is the lower-right cell), repeating the recurrence formulae to decide in which direction to move next. As the recurrence returns, the alignment is output. We provide a stack based method for traceback in Algorithm 2.2 and Algorithm 2.3.

The time complexity of the traceback part (Algorithm 2.2 and Algorithm 2.3) is $O(|S_1| + |S_2|)$, and the space complexity is $O(|S_1||S_2|)$.

Therefore, the time complexity of the whole dynamic programming solution is $O(|S_1||S_2|)$, and the space complexity is $O(|S_1||S_2|)$.

Algorithm 2.2 *Sequence-Traceback*(S_1, S_2, $M(0..|S_1|; 0..|S_2|)$)

Input: Two sequences S_1 and S_2 with $n = |S_1|$ and $m = |S_2|$, and alignment score matrix $M(0..|S_1|; 0..|S_2|)$ as computed by Algorithm 2.1.

Output: A stack *stack* containing ordered pairs of sequence indices in an alignment (S'_1, S'_2) with the minimum score as computed by Algorithm 2.1.

```
1: $i \leftarrow n$
2: $j \leftarrow m$
3: while $i > 0$ and $j > 0$ do
4:   if $A(i,j) = A(i-1,j) + \gamma(i,0)$ then
5:     $push\_stack(i, -1)$
6:     $i \leftarrow i - 1$
7:   else if $A(i,j) = A(i,j-1) + \gamma(0,j)$ then
8:     $push\_stack(-1, j)$
9:     $j \leftarrow j - 1$
10:  else // $A(i,j) = A(i-1,j-1) + \gamma(i,j)$
11:    $push\_stack(i, j)$
12:    $i \leftarrow i - 1$
13:    $j \leftarrow j - 1$
14:  end if
15: end while
16: while $i > 0$ do
17:   $push\_stack(i, -1)$
18:   $i \leftarrow i - 1$
19: end while
20: while $j > 0$ do
21:   $push\_stack(-1, j)$
22:   $j \leftarrow j - 1$
23: end while
```

Algorithm 2.3 *Sequence-Align*(S_1, S_2, $M(0..|S_1|;0..|S_2|)$)

```
Input: Two sequences $S_1$ and $S_2$ with $n = |S_1|$ and $m = |S_2|$, and alignment score
    matrix $M(0..|S_1|;0..|S_2|)$ as computed by Algorithm 2.1.
Output: An optimal alignment Align between $S_1$ and $S_2$.
 1: Sequence-Traceback($S_1$, $S_2$, $M(0..|S_1|;0..|S_2|)$) // Algorithm 2.2
 2: while stack is not empty do
 3:   $(i,j) \leftarrow$ pop_stack
 4:   if $i > 0$ and $j = -1$ then
 5:     append $(S_1[i], '-')$ to the alignment
 6:   else if $i = -1$ and $j > 0$ then
 7:     append $('-', S_2[j])$ to the alignment
 8:   else // $i > 0$ and $j > 0$
 9:     append $(S_1[i], S_2[j])$ to the alignment
10:   end if
11: end while
```

2.1.3.4 An Example

We now consider a simple example. We want to compute the alignment between two sequences $S_1 = AAUAAGU$ and $S_2 = AUAACAU$. The score scheme used is shown in Table 2.1. The value of entry (a, b) of the table $(a, b \in \{A, C, G, U, -\})$ is the cost of substituting a with b if $a \neq '-'$ and $b \neq '-'$, or the cost of deleting a if $a \neq '-'$ and $b = '-'$, or the cost of inserting b if $a = '-'$ and $b \neq '-'$. (Notice that the entry $('-','-')$ does not exist since we cannot align $'-'$ to $'-'$.)

	A	C	G	U	$-$
A	0	1	1	1	1
C	1	0	1	1	1
G	1	1	0	1	1
U	1	1	1	0	1
$-$	1	1	1	1	

Table 2.1: A simple score scheme

Applying Algorithm 2.1 to the example, we obtain the alignment score matrix which is shown in Table 2.2. The optimal alignment score is 3 as the lower-right cell shows.

Applying Algorithm 2.3 to Table 2.2, we obtain an optimal alignment between

	$-$	A	U	A	A	C	A	U
$-$	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
A	2	1	1	1	2	3	4	5
U	3	2	1	2	2	3	4	4
A	4	3	2	1	2	3	3	4
A	5	4	3	2	1	2	3	4
G	6	5	4	3	2	2	3	4
U	7	6	5	4	3	3	3	3

Table 2.2: Computation matrix of alignment between two sequences $AAUAAGU$ and $AUAACAU$

S_1 and S_2. In Table 2.3, the path marked by the asterisk signs corresponds to the optimal alignment. The alignment is shown as follows.

$$\begin{array}{cccccccc} A & A & U & A & A & G & - & U \\ A & - & U & A & A & C & A & U \end{array}$$

	$-$	A	U	A	A	C	A	U
$-$	0	1	2	3	4	5	6	7
A	1	0*	1	2	3	4	5	6
A	2	1*	1	1	2	3	4	5
U	3	2	1*	2	2	3	4	4
A	4	3	2	1*	2	3	3	4
A	5	4	3	2	1*	2	3	4
G	6	5	4	3	2	2*	3*	4
U	7	6	5	4	3	3	3	3*

Table 2.3: Trace back to produce an optimal alignment between two sequences $AAUAAGU$ and $AUAACAU$

2.2 Gap

A gap in an alignment (S'_1, S'_2) is a consecutive subsequence of $'-'$'s in either S'_1 or S'_2 with maximal length. More formally, $[i \cdots j]$ is a gap in (S'_1, S'_2) if either $S'_1[k] = '-'$ for $i \leq k \leq j$, $S'_1[i-1] \neq '-'$, $S'_1[j+1] \neq '-'$, or $S'_2[k] = '-'$ for $i \leq k \leq j$, $S'_2[i-1] \neq '-'$, $S'_2[j+1] \neq '-'$.

2.2.1 Gap Penalty Models

When mutations take place in nature, it is widely believed that the occurrence of a gap with k consecutive spaces is more probable than k separated spaces. This is because a gap may be due to a mutation which deletes several consecutive bases in one single event whereas separated spaces are more likely due to several different events. A single event is more common to happen than several different events. Therefore, it is sometimes required to weight the cost of deletion (or insertion) of a number of consecutive bases differently from summing the costs of single deletions (or insertions).

Now we introduce a general gap penalty model. Let $g_1(i_1, i)$ be the cost for deletion of characters from $S_1[i_1]$ to $S_1[i]$ of S_1, and $g_2(j_1, j)$ be the cost for insertion of characters from $S_2[j_1]$ to $S_2[j]$ of S_2. It is reasonable that $g_1(i_1, i) \leq \sum_{k=i_1}^{i} g_1(k, k)$ and $g_2(j_1, j) \leq \sum_{l=j_1}^{j} g_2(l, l)$.

We now consider how to compute the optimal alignment cost $A(i, j)$ between $S_1[1, i]$ and $S_2[1, j]$ under this model. Similar to Lemmas 2.1.1 to 2.1.4, we can get following lemmas.

Lemma 2.2.1

$$A(0, 0) = 0 \tag{2.12}$$

Proof: Similar to Lemma 2.1.1. □

Lemma 2.2.2 *For* $i > 0$,

$$A(i, 0) = g_1(1, i) \tag{2.13}$$

Proof: It is obvious that each element in $S_1[1, i]$ is aligned to $'-'$. Thus subsequence $S_1[1, i]$ is deleted. Hence we have $A(i, 0) = g_1(1, i)$. □

Lemma 2.2.3 *For* $j > 0$,

$$A(0, j) = g_2(1, j) \tag{2.14}$$

Proof: Similar to Lemma 2.2.2. □

Lemma 2.2.4 *For $i > 0$ and $j > 0$,*

$$A(i,j) = \min \begin{cases} \min_{1\le k\le i}\{A(i-k,j) + g_1(i-k+1,i)\} \\ \min_{1\le l\le j}\{A(i,j-l) + g_2(j-l+1,j)\} \\ A(i-1,j-1) + \gamma(i,j) \end{cases} \tag{2.15}$$

Proof: There are exactly the following cases.

(1) Each element in $S_1[i-k+1,i]$ $(1 \le k \le i)$ is aligned to '$-$'. Thus subsequence $S_1[i-k+1,i]$ is deleted, and $S_1[1,i-k]$ is aligned to $S_2[1,j]$. Hence the $A(i-k,j) + g_1(i-k+1,i)$ item. We need to iterate over all possible instances of k. Thus we have the $\min_{1\le k\le i}\{A(i-k,j) + g_1(i-k+1,i)\}$ item.

(2) Each element in $S_2[j-l+1,j]$ $(1 \le l \le j)$ is aligned to '$-$'. Thus subsequence $S_2[j-l+1,j]$ is inserted, and $S_1[1,i]$ is aligned to $S_2[1,j-l]$. Hence the $A(i,j-l) + g_2(j-l+1,j)$ item. We need to iterate over all possible instances of l. Thus we have the $\min_{1\le l\le j}\{A(i,j-l) + g_2(j-l+1,j)\}$ item.

(3) $S_1[i]$ is aligned to $S_2[j]$. Thus $S_1[1,i-1]$ is aligned to $S_2[1,j-1]$. Hence the $A(i-1,j-1) + \gamma(i,j)$[1] item.

Therefore we take the minimum of the three cases and get the above recursion. □

From Lemmas 2.2.1 to 2.2.4, we can compute $Align(S_1,S_2) = A(|S_1|,|S_2|)$ using a bottom-up approach. It is not hard to see that the computation time is $1 + |S_1| + |S_2| + \sum_{i=1}^{|S_1|}\sum_{j=1}^{|S_2|}(i+j+1) = O(|S_1|^2|S_2| + |S_2|^2|S_1|)$.

It is possible to reduce the running time for some specific gap penalty functions. When $g_1(i_1,i)$ and $g_2(j_1,j)$ are linear (affine), the running time is $O(|S_1||S_2|)$. The linear gap penalty model has already been used in Section 2.1. Now we introduce the affine gap penalty model.

The affine gap penalty model has long been used in sequence alignment [7]. For each gap in an alignment, in addition to the insertion/deletion costs, we will assign a constant, *gap_cost*, as the gap initiation cost. This means that longer

[1]Recall that $\gamma(i,j)$ is the cost of aligning $S_1[i]$ to $S_2[j]$.

gaps are preferred since for a longer gap the additional cost distributed to each base is relatively small. The corresponding gap penalty functions are $g_1(i_1, i) = gap\_cost + \sum_{k=i_1}^{i} g_1(k,k)$ and $g_2(j_1, j) = gap\_cost + \sum_{l=j_1}^{j} g_2(l,l)$.

Under the affine gap penalty model, the cost of an alignment (S_1', S_2') is defined as follows, where $\#gap$ is the number of gaps in (S_1', S_2').

$$\begin{aligned} cost((S_1', S_2')) = \quad & gap\_cost \times \#gap \\ & + \sum_{i \in M} \Gamma(S_1'[i] \rightarrow S_2'[i]) \\ & + \sum_{i \in D} \Gamma(S_1'[i] \rightarrow \lambda) \\ & + \sum_{i \in I} \Gamma(\lambda \rightarrow S_2'[i]). \end{aligned} \tag{2.16}$$

2.2.2 Computing Optimal Alignment with Affine Gap Penalty

In this section, we will discuss how to compute the optimal alignment between two sequences S_1 and S_2 under the affine gap penalty model.

2.2.2.1 Property of Optimal Alignments

We now consider the optimal alignment between $S_1[1,i]$ and $S_2[1,j]$. We use $A(i,j)$ to represent the optimal alignment cost between $S_1[1,i]$ and $S_2[1,j]$. We use $D(i,j)$ to represent the optimal alignment cost such that $S_1[i]$ is aligned to '$-$'. We use $I(i,j)$ to represent the optimal alignment cost such that $S_2[j]$ is aligned to '$-$'. $\gamma(i,0)$ is the cost of deleting character $S_1[i]$, $\gamma(0,j)$ is the cost of inserting character $S_2[j]$, and $\gamma(i,j)$ is the cost of aligning $S_1[i]$ to $S_2[j]$. The following lemmas will show how to compute $A(i,j)$.

Lemma 2.2.5

$$A(0,0) = 0 \tag{2.17}$$

$$D(0,0) = gap\_cost \tag{2.18}$$

$$I(0,0) = gap\_cost \tag{2.19}$$

Proof: For $A(0,0)$, consider $A(1,1)$. If the optimal alignment results from aligning $S_1[1]$ to $S_2[1]$, then we only need to account for the cost for aligning $S_1[1]$ to $S_2[1]$. Hence we may set $A(0,0) = 0$.

For $D(0,0)$, consider $D(1,0)$ by which $S_1[1]$ is aligned to $'-'$. Aligning $S_1[1]$ to $'-'$ opens a gap, so we need to charge gap opening penalty for it. Hence we may set $D(0,0) = gap\_cost$.

Similarly, we can set $I(0,0) = gap\_cost$. □

Lemma 2.2.6 *For $i > 0$,*

$$D(i,0) = D(i-1,0) + \gamma(i,0) \tag{2.20}$$

$$A(i,0) = D(i,0) \tag{2.21}$$

$$I(i,0) = D(i,0) + gap\_cost \tag{2.22}$$

For $j > 0$,

$$I(0,j) = I(0,j-1) + \gamma(0,j) \tag{2.23}$$

$$A(0,j) = I(0,j) \tag{2.24}$$

$$D(0,j) = I(0,j) + gap\_cost \tag{2.25}$$

Proof: For $D(i,0)$, by definition $S_1[i]$ is aligned to $'-'$, hence we have the $\gamma(i,0)$ term, and $S_1[1,i-1]$ is aligned to $\emptyset$. That is, each element in $S_1[1,i-1]$ is aligned to $'-'$, by which we know that $S_1[i-1]$ is aligned to $'-'$. Hence we have $D(i,0) = D(i-1,0) + \gamma(i,0)$.

For $A(i,0)$, this is the optimal alignment between $S_1[1,i]$ and $\emptyset$. Thus each element in $S_1[1,i]$ is aligned to $'-'$, by which we know that $S_1[i]$ is aligned to $'-'$. Hence we have $A(i,0) = D(i,0)$.

For $I(i,0)$, consider $I(i,1)$, the optimal alignment between $S_1[1,i]$ and $S_2[1,1]$ that ends with $S_2[1]$ aligned to $'-'$. Thus $S_1[1,i]$ is aligned to $\emptyset$. That is, each element in $S_1[1,i]$ is aligned to $'-'$, by which we know that $S_1[i]$ is aligned to $'-'$. Aligning $S_2[1]$ to $'-'$ opens a gap, so we need to charge gap opening penalty for it. Hence we have $I(i,0) = D(i,0) + \mathit{gap\_cost}$.

Similarly, we can obtain other three formulas. □

Lemma 2.2.7 *For $i > 0$ and $j > 0$,*

$$D(i,j) = \min \begin{cases} D(i-1,j) + \gamma(i,0) \\ A(i-1,j) + \gamma(i,0) + \mathit{gap\_cost} \end{cases} \tag{2.26}$$

Proof: We use $M(i,j)$ to represent the optimal alignment cost such that $S_1[i]$ is aligned to $S_2[j]$. Then

$$A(i,j) = \min \begin{cases} D(i,j) \\ I(i,j) \\ M(i,j) \end{cases}$$

According to the definition of $D(i,j)$, $S_1[i]$ is aligned to $'-'$, hence $\gamma(i,0)$. We consider $S_1[i-1]$ and $S_2[j]$, there are exactly the following cases.

(1) $S_1[i-1]$ is aligned to $'-'$. $D(i,j)$ is from $D(i-1,j)$, then aligning $S_1[i]$ to $'-'$ does not open a gap. Therefore there is no gap opening penalty.

(2) $S_2[j]$ is aligned to $'-'$. $D(i,j)$ is from $I(i-1,j)$, then aligning $S_1[i]$ to $'-'$ opens a gap. Therefore there is a gap opening penalty.

(3) $S_1[i-1]$ is aligned to $S_2[j]$. $D(i,j)$ is from $M(i-1,j)$, then aligning $S_1[i]$ to $'-'$ opens a gap. Therefore there is a gap opening penalty.

So we have the following recursion.

$$D(i,j) = \min \begin{cases} D(i-1,j) + \gamma(i,0) \\ I(i-1,j) + \gamma(i,0) + \mathit{gap\_cost} \\ M(i-1,j) + \gamma(i,0) + \mathit{gap\_cost} \end{cases}$$

$$= \min \begin{cases} D(i-1,j) + \gamma(i,0) \\ D(i-1,j) + \gamma(i,0) + \mathit{gap\_cost} \\ I(i-1,j) + \gamma(i,0) + \mathit{gap\_cost} \\ M(i-1,j) + \gamma(i,0) + \mathit{gap\_cost} \end{cases}$$

$$= \min \begin{cases} D(i-1,j) + \gamma(i,0) \\ A(i-1,j) + \gamma(i,0) + \mathit{gap\_cost} \end{cases}$$

□

Lemma 2.2.8 *For $i > 0$ and $j > 0$,*

$$I(i,j) = \min \begin{cases} I(i,j-1) + \gamma(0,j) \\ A(i,j-1) + \gamma(0,j) + \mathit{gap\_cost} \end{cases} \tag{2.27}$$

Proof: Similar to Lemma 2.2.7. □

Lemma 2.2.9 *For $i > 0$ and $j > 0$,*

$$A(i,j) = \min \begin{cases} D(i,j) \\ I(i,j) \\ A(i-1,j-1) + \gamma(i,j) \end{cases} \tag{2.28}$$

Proof: Consider $S_1[i]$ and $S_2[j]$. There are exactly the following cases.

(1) $S_1[i]$ is aligned to '$-$'. Hence the $D(i,j)$ item.

(2) $S_2[j]$ is aligned to $'-'$. Hence the $I(i, j)$ item.

(3) $S_1[i]$ is aligned to $S_2[j]$. Thus $S_1[1, i-1]$ is aligned to $S_2[1, j-1]$. Hence the $A(i-1, j-1) + \gamma(i, j)$ item.

Therefore we take the minimum of the three cases and get the above recursion. □

2.2.2.2 Algorithm

From Lemmas 2.2.5 to 2.2.9, we can compute $Align(S_1, S_2) = A(|S_1|, |S_2|)$ using a bottom-up approach. Algorithm 2.4 shows how to compute optimal alignment score with affine gap penalty.

Algorithm 2.4 *Sequence-Distance-Gap*(S_1, S_2)

```
Input: Two sequences $S_1$ and $S_2$ with $n = |S_1|$ and $m = |S_2|$.
Output: Alignment score matrices $M_{A,D,I}(0..|S_1|; 0..|S_2|)$.
 1: compute $A(0,0)$, $D(0,0)$ and $I(0,0)$ as in Lemmas 2.2.5
 2: for $i \leftarrow 1$ to $n$ do
 3:    compute $D(i,0)$, $A(i,0)$ and $I(i,0)$ as in Lemmas 2.2.6
 4: end for
 5: for $j \leftarrow 1$ to $m$ do
 6:    $I(0,j)$, $A(0,j)$ and $D(0,j)$ as in Lemmas 2.2.6
 7: end for
 8: for $i \leftarrow 1$ to $n$ do
 9:    for $j \leftarrow 1$ to $m$ do
10:       compute $D(i,j)$, $I(i,j)$ and $A(i,j)$ as in Lemmas 2.2.7, Lemmas 2.2.8 and
          Lemmas 2.2.9, respectively
11:    end for
12: end for
```

In the implementation, we need three matrices instead of one in Section 2.1.3.3. These matrices are $M_A(0..|S_1|; 0..|S_2|)$, $M_D(0..|S_1|; 0..|S_2|)$, and $M_I(0..|S_1|; 0..|S_2|)$, each with size $(|S_1| + 1)(|S_2| + 1)$. We use $M_{A,D,I}(0..|S_1|; 0..|S_2|)$ to represent these three matrices in the text to save space.

The time complexity of Algorithm 2.4 is $O(|S_1||S_2|)$, and the space complexity is $O(|S_1||S_2|)$.

To produce an optimal alignment, we *trace back* among the three matrices. We provide a stack based method for traceback in Algorithm 2.5 to 2.7. The variable t

used in algorithm *Sequence-Traceback-Gap* (see Algorithm 2.5 and 2.6) is matrix type where $t \in \{A, D, I\}$.

The time complexity of the traceback part (Algorithm 2.5 to 2.7) is $O(|S_1| + |S_2|)$, and the space complexity is $O(|S_1||S_2|)$.

Therefore, the time complexity of the whole algorithm for computing optimal alignment with affine gap penalty is $O(|S_1||S_2|)$, and the space complexity is $O(|S_1||S_2|)$.

Algorithm 2.5 *Sequence-Traceback-Gap*(S_1, S_2, $M_{A,D,I}(0..|S_1|;0..|S_2|)$) Part 1

Input: Two sequences S_1 and S_2 with $n = |S_1|$ and $m = |S_2|$, and alignment score matrices $M_{A,D,I}(0..|S_1|;0..|S_2|)$ as computed by Algorithm 2.4.

Output: A stack *stack* containing ordered pairs of sequence indices in an alignment (S_1', S_2') with the minimum score as computed by Algorithm 2.4.

```
1:  $t \leftarrow A$
2:  $i \leftarrow n$
3:  $j \leftarrow m$
4:  while $i > 0$ and $j > 0$ do
5:     if $t = A$ then
6:        if $A(i,j) = D(i,j)$ then
7:           $t \leftarrow D$
8:        else if $A(i,j) = I(i,j)$ then
9:           $t \leftarrow I$
10:       else // $A(i,j) = A(i-1,j-1) + \gamma(i,j)$
11:          $push\_stack(i,j)$
12:          $i \leftarrow i-1$
13:          $j \leftarrow j-1$
14:       end if
15:    else if $t = D$ then
16:       if $D(i,j) = D(i-1,j) + \gamma(i,0)$ then
17:          $push\_stack(i,-1)$
18:          $i \leftarrow i-1$
19:       else // $D(i,j) = A(i-1,j) + \gamma(i,0) + gap\_cost$
20:          $t \leftarrow A$
21:          $push\_stack(i,-1)$
22:          $i \leftarrow i-1$
23:       end if
24:    else // $t = I$
25:       if $I(i,j) = I(i,j-1) + \gamma(0,j)$ then
26:          $push\_stack(-1,j)$
27:          $j \leftarrow j-1$
28:       else // $I(i,j) = A(i,j-1) + \gamma(0,j) + gap\_cost$
29:          $t \leftarrow A$
30:          $push\_stack(-1,j)$
31:          $j \leftarrow j-1$
32:       end if
33:    end if
34: end while
35: // This algorithm is too long to fit on one page, we need to break here! To be
    continued in Algorithm 2.6 on next page.
```

Algorithm 2.6 *Sequence-Traceback-Gap*(S_1, S_2, $M_{A,D,I}(0..|S_1|; 0..|S_2|)$) Part 2

```
36: // Continued from Algorithm 2.5 on last page.
37: while $i > 0$ do
38:   $push\_stack(i, -1)$
39:   $i \leftarrow i - 1$
40: end while
41: while $j > 0$ do
42:   $push\_stack(-1, j)$
43:   $j \leftarrow j - 1$
44: end while
```

Algorithm 2.7 *Sequence-Align-Gap*(S_1, S_2, $M_{A,D,I}(0..|S_1|; 0..|S_2|)$)

Input: Two sequences S_1 and S_2 with $n = |S_1|$ and $m = |S_2|$, and alignment score matrices $M_{A,D,I}(0..|S_1|; 0..|S_2|)$ as computed by Algorithm 2.4.

Output: An optimal alignment *Align* between S_1 and S_2.

```
1: Sequence-Traceback-Gap($S_1$, $S_2$, $M_{A,D,I}(0..|S_1|; 0..|S_2|)$) // Algorithm 2.5 and 2.6
2: while stack is not empty do
3:   $(i, j) \leftarrow pop\_stack$
4:   if $i > 0$ and $j = -1$ then
5:     append $(S_1[i], '-')$ to the alignment
6:   else if $i = -1$ and $j > 0$ then
7:     append $('-', S_2[j])$ to the alignment
8:   else // $i > 0$ and $j > 0$
9:     append $(S_1[i], S_2[j])$ to the alignment
10:  end if
11: end while
```

Chapter 3

A Review of RNA Structural Alignment

We have discussed alignment at sequence level in Chapter 2. The main theme in this thesis concerns alignment at the structural level in which we also need to align the base pairs. In this chapter, we consider the problem of structural alignment between two RNA structures. More specifically, we focus on alignment between two RNA tertiary structures.

The problem of aligning structures is similar to the problem of aligning sequences but much more complicated due to the presence of the base pairs. The presence of base pairs also makes the computation more resources (time and space) consuming.

In this chapter, we first introduce the RNA alignment model and the hardness results of the RNA structural alignment problem. Then we discuss Wang and Zhang's RNA alignment algorithm and Möhl *et al.*'s RNA alignment algorithm which serve as bases of our algorithm in detail.

3.1 RNA Alignment Models

An RNA structure is represented by $R(P)$, where R is a sequence of nucleotides with $R[i]$ representing the i-th nucleotide, and $P \subset \{1, 2, \cdots, |R|\}^2$ is a set of arcs of which each element (i, j), $i < j$, represents the bond between the two bases of a base pair $(R[i], R[j])$ in R. We use $R[i, j]$ to represent the subsequence of R from $R[i]$ to $R[j]$, and $|R|$ to represent the length of R. We assume that base pairs in $R(P)$ do not share participating bases. Formally for any (i_1, j_1) and (i_2, j_2) in P, $j_1 \neq i_2$, $i_1 \neq j_2$, and $i_1 = i_2$ if and only if $j_1 = j_2$. The left end l and right end r of an arc $p = (l, r) \in P$ are denoted by p^L and p^R, respectively.

Let p and p' be two arcs in $R(P)$. Their corresponding base pairs are $bp = (R[p^L], R[p^R])$ and $bp' = (R[p'^L], R[p'^R])$, respectively. We define the relation between p and p' (bp and bp') as follows. We say that p (bp) is *before* p' (bp') if $p^R < p'^L$; alternatively, we say that p' (bp') is *after* p (bp) (see Figure 3.1). We say that p (bp) is *inside* p' (bp') if $p'^L < p^L < p^R < p'^R$; alternatively, we say that p' (bp') is *outside* p (bp) (see Figure 3.2). We say that p (bp) is *crossed by* p' (bp') if $p^L < p'^L < p^R < p'^R$ or $p'^L < p^L < p'^R < p^R$; in the first case, p (bp) is *right crossed by* p' (bp'), in the second case p (bp) is *left crossed by* p' (bp') (see Figure 3.3).

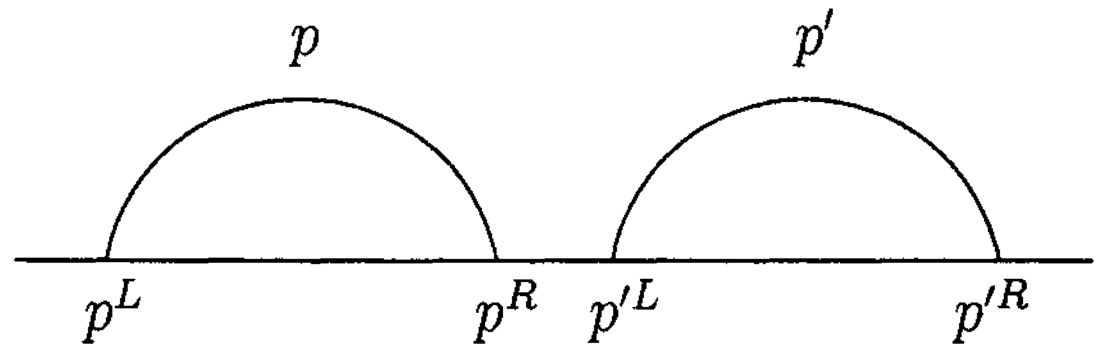


Figure 3.1: p is before p'; p' is after p

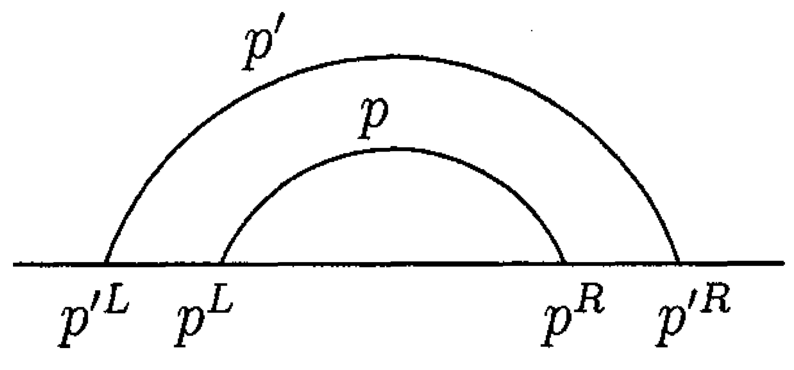


Figure 3.2: p is inside p'; p' is outside p

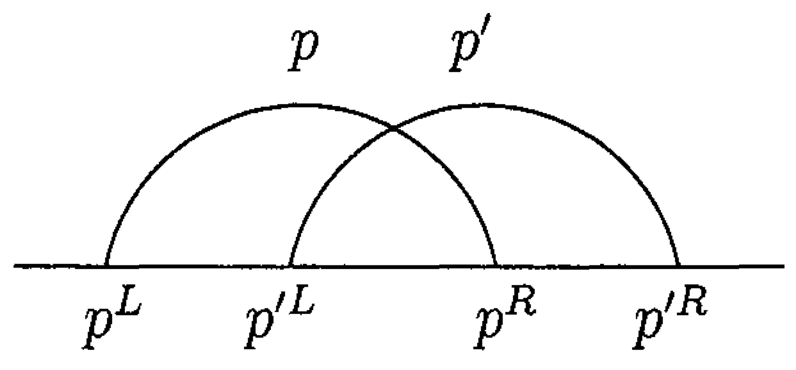


Figure 3.3: p is right crossed by p'; p' is left crossed by p

An arc p (a base pair bp) is called *crossing* if it is crossed by an arc p' (a base pair bp'). If p (bp) is right crossed by p' (bp'), we say that p (bp) is *right crossing*; if p (bp) is left crossed by p' (bp'), we say that p (bp) is *left crossing*. An arc p (a base pair bp) is called *non-crossing* if it is not crossed by any arc p' (base pair bp'). An RNA structure $R(P)$ containing crossing arcs (base pairs) is called *crossing*, otherwise *non-crossing*.

A *set of tertiary arcs* of an RNA structure $R(P)$ is a subset of crossing arcs $P_{ter} \subseteq P$ which satisfies the condition that for any two arcs $p, p' \in P - P_{ter}$, p and p' do not cross. We call $P_{sec} = P - P_{ter}$ a *set of secondary arcs*. The corresponding base pairs of secondary arcs and tertiary arcs are called *secondary base pairs* and *tertiary base pairs*, respectively.

For an RNA structure $R(P)$, we define $p_r(\ )$ as follows.

$$p_r(i) = \begin{cases} j & \text{if } \exists j : (i,j) \in P \text{ or } (j,i) \in P \\ i & \text{otherwise} \end{cases} \tag{3.1}$$

By this definition, $p_r(i) \neq i$ if and only if $R[i]$ is a base in a base pair of $R(P)$, and $p_r(i) = i$ if and only if $R[i]$ is an unpaired base of $R(P)$. If $p_r(i) \neq i$, then $p_r(i)$ is the base paired with base i. When there is no confusion, we use R, instead of $R(P)$, to represent an RNA structure assuming that there is an associated function $p_r(\ )$.

Given two RNA structure R_1 and R_2 which are over the four-letter alphabet $\Sigma = \{A, C, G, U\}$, the alignment of R_1 and R_2 is represented by (R_1', R_2') satisfying the following conditions:

- R'_1 is R_1 with some new symbol $'-'$ inserted and R'_2 is R_2 with some new symbol $'-'$ inserted such that $|R'_1| = |R'_2|$.
- $\forall i \in \{1, \ldots, |R'_1|\}$, at least one of $R'_1[i]$ and $R'_2[i]$ cannot be $'-'$.

For an alignment (R'_1, R'_2), we use $(R'_1, R'_2)[i_1, i_2; j_1, i_2]$ to represent the *subalignment* which consists of subsequence of R'_1 from the i_1-th non-space character to the i_2-th non-space character and subsequence of R'_2 from the j_1-th non-space character to the j_2-th non-space character.

An *arc pair* is a pair of arcs $a = (p_1, p_2) \in P_1 \times P_2$. We call $a = (p_1, p_2)$ *realized by* (R'_1, R'_2) if and only if p_1 and p_2 are matched by (R'_1, R'_2). The *set* $OA((R'_1, R'_2)[i_1, i_2; j_1, i_2])$ *of open arc pairs of a subalignment* $(R'_1, R'_2)[i_1, i_2; j_1, i_2]$ *in* (R'_1, R'_2) is the set of arc pairs (p_1, p_2) that are realized by (R'_1, R'_2) and where $p_1^L < i_1 \le p_1^R \le i_2$ and $p_2^L < j_1 \le p_2^R \le j_2$ or $i_1 \le p_1^L \le i_2 < p_1^R$ and $j_1 \le p_2^L \le j_2 < p_2^R$. (That means, one ends of the arcs p_1 and p_2 are matched inside the subalignment $(R'_1, R'_2)[i_1, i_2; j_1, i_2]$, and the other ends of p_1 and p_2 are matched outside of the subalignment $(R'_1, R'_2)[i_1, i_2; j_1, i_2]$.) In the example shown in Figure 3.4, arc pairs (1, I), (2, II), (3, III) and (4, IV) are realized by the alignment. The region enclosed by the rectangle is one subalignment. According to the previous definition, the set of open arc pairs of this subalignment is $\{(1, \text{I}), (3, \text{III}), (4, \text{IV})\}$.

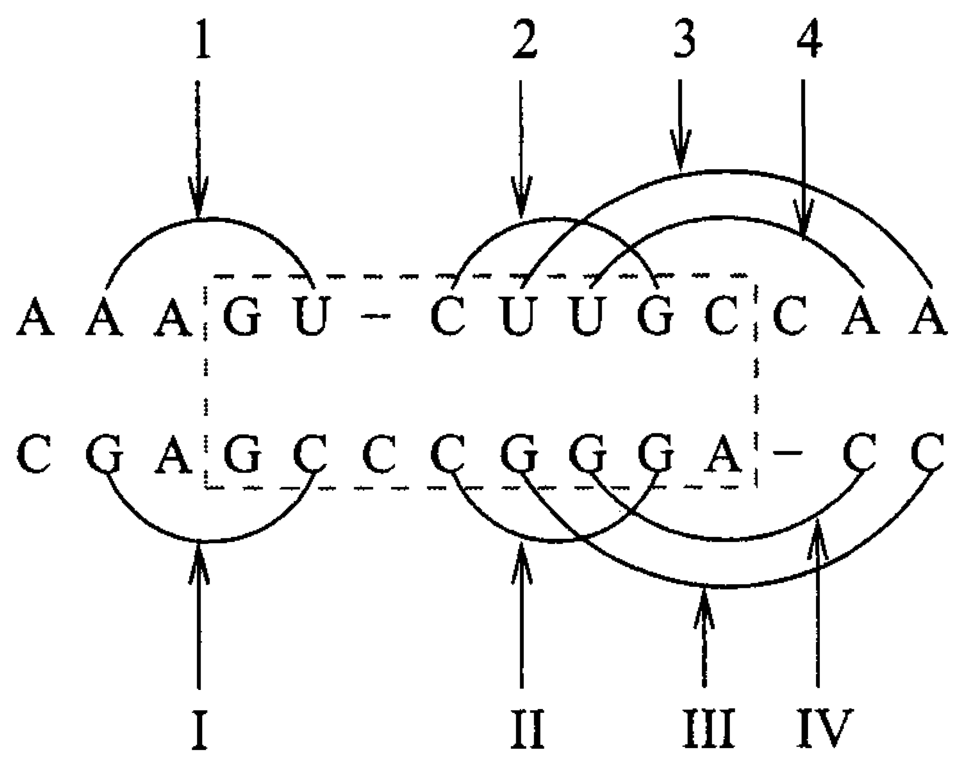


Figure 3.4: An example of arc pairs

We define the left and right end point of an arc pair (p_1, p_2) as $\nwarrow (p_1, p_2) = (p_1^L, p_2^L)$

and $\searrow (p_1, p_2) = (p_1^R, p_2^R)$, respectively. On those points we consider the partial order $\prec$ defined as $(x_1, y_1) \prec (x_2, y_2)$ iff $x_1 < x_2$ and $y_1 < y_2$. Two arc pairs a and a' cross, iff $\nwarrow a \prec \nwarrow a' \prec \searrow a \prec \searrow a'$ or $\nwarrow a' \prec \nwarrow a \prec \searrow a' \prec \searrow a$. In the example shown in Figure 3.5, arc pairs (1, I) and (2, II) cross; (1, II) and (2, III) cross; (2, I) and (3, II) cross; (2, II) and (3, III) cross.

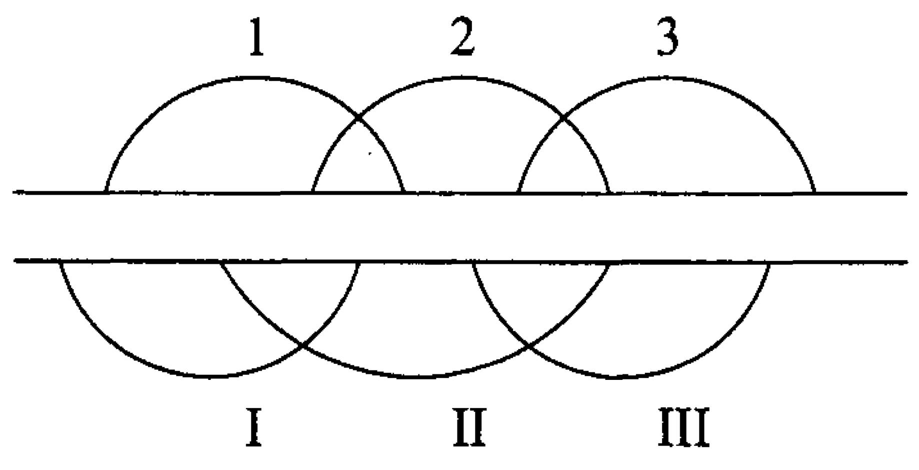


Figure 3.5: An example of crossing arc pairs

Following the tradition in sequence comparison [14, 18], we define three edit operations, *substitute*, *delete*, and *insert*, on RNA structures. For a given RNA structure R, each operation can be applied to either a base pair or an unpaired base. To substitute a base pair is to replace one base pair with another. This means that at the sequence level, two bases may be changed at the same time. To delete a base pair is to remove the base pair. At the sequence level, this means to delete two bases at the same time. To insert a base pair is to insert a new base pair. At the sequence level, this means to insert two bases at the same time.

In addition to the edit operations of *insertion*, *deletion*, and *substitution*, we now consider two more operation: *base-pair bond breaking* and *base-pair altering*. *Base-pair bond breaking* operation can be applied to a base pair, causing the bond between the two bases of the pair to break and the base pair to become two unpaired bases. *Base-pair altering* operation can be applied to a base pair, causing the bond between the two bases of the pair to break and one base of the base pair to be deleted, leaving the other base unpaired. *Base-pair bond breaking* and *base-pair altering* can also be called *arc breaking* and *arc altering*, respectively.

Now we give a formal presentation. Let $\Sigma = \{A, C, G, U\}$, $\Sigma_1 = \Sigma \times \Sigma$, $\Sigma_2 =$

$\Sigma \times \{\lambda\}$, and $\Sigma_3 = \{\lambda\} \times \Sigma$, where λ is the null label. We represent an edit operation as $a \rightarrow b$, where a and b are either λ, or labels from Σ or Σ_1 or Σ_2 or Σ_3.

We call $a \rightarrow b$ a base substitute operation if $a, b \in \Sigma$ (if $a = b$, it is a base match operation; else it is a base mismatch operation); a base delete operation if $a \in \Sigma$ and $b = \lambda$; and a base insert operation if $b \in \Sigma$ and $a = \lambda$.

We say $a = (a_1, a_2)$ is a base pair if $a \in \Sigma_1$, a_1 and a_2 are bases in the same base pair. We call $a \rightarrow b$ a base-pair substitute operation if both a and b are base pairs (if $a = b$, it is a base-pair match operation; else it is a base-pair mismatch operation); a base-pair delete operation if a is a base pair, and $b = \lambda$; a base-pair insert operation if b is a base pair, and $a = \lambda$; a base-pair bond breaking operation if $a, b \in \Sigma_1$, and one of a and b is a base pair and the other one is not a base pair; a base-pair altering operation if a is a base pair and $b \in \Sigma_2$, or a is a base pair and $b \in \Sigma_3$, or $a \in \Sigma_2$ and b is a base pair, or $a \in \Sigma_3$ and b is a base pair.

Figure 3.6 gives an illustration of RNA structure alignment with edit operations.

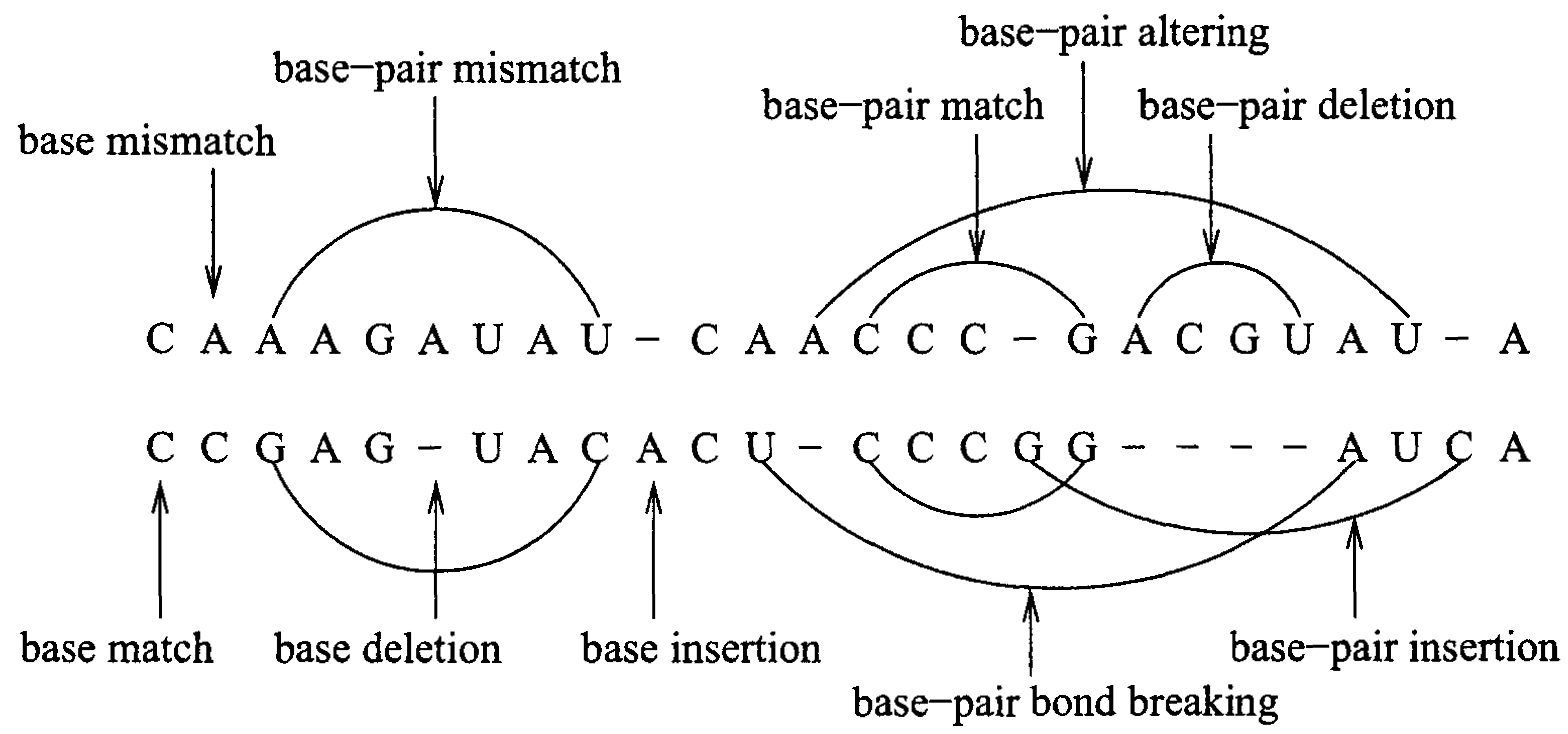


Figure 3.6: RNA structure alignment with edit operations

Let Γ be a cost function which assigns to each edit operation $a \rightarrow b$ a nonnegative real number $\Gamma(a \rightarrow b)$. We constrain Γ to be a distance metric. That is, (1) $\Gamma(a \rightarrow b) \geq 0$, $\Gamma(a \rightarrow a) = 0$, (2) $\Gamma(a \rightarrow b) = \Gamma(b \rightarrow a)$, and (3) $\Gamma(a \rightarrow c) \leq \Gamma(a \rightarrow$

$b) + \Gamma(b \rightarrow c)$. An item $\Gamma(a \rightarrow b)$ where $a, b \in \Sigma_1$, a is a base pair and b is not a base pair represents the base-pair bond breaking cost of a. b does not affect this cost. To distinguish this item from $\Gamma(a \rightarrow b)$ where both a and b are base pairs which represent the base-pair substitution cost, we use $\Gamma_b(a)$ to replace this item. Analogously, we use $\Gamma_b(b)$ to denote the base-pair bond breaking cost of the base pair b, replacing $\Gamma(a \rightarrow b)$ where $a, b \in \Sigma_1$, a is not a base pair and b is a base pair.

Given an alignment (R'_1, R'_2), we define single base substitution SM, single base deletion SD, single base insertion SI, base pair substitution PM, base pair deletion PD, base pair insertion PI, case 1 of base pair bond breaking PB_1, case 2 of base pair bond breaking PB_2, case 1 of base pair altering PA_1, case 2 of base pair altering PA_2, case 3 of base pair altering PA_3, and case 4 of base pair altering PA_4 as follows.

SM $= \{\ i \mid R'_1[i]$ and $R'_2[i]$ are unpaired bases or bases in the base pairs that have undergone base-pair bond breakings or base-pair alterings in R_1 and $R_2\}$.

SD $= \{\ i \mid R'_1[i]$ is an unpaired bases in R_1 and $R'_2[i] = '-'\}$.

SI $= \{\ i \mid R'_2[i]$ is an unpaired bases in R_2 and $R'_1[i] = '-'\}$.

PM $= \{\ (i,j) \mid (R'_1[i], R'_1[j])$ and $(R'_2[i], R'_2[j])$ are base pairs in R_1 and $R_2\}$.

PD $= \{\ (i,j) \mid (R'_1[i], R'_1[j])$ is a base pair in R_1, and $R'_2[i] = R'_2[j] = '-'\}$.

PI $= \{\ (i,j) \mid (R'_2[i], R'_2[j])$ is a base pair in R_2, and $R'_1[i] = R'_1[j] = '-'\}$.

$PB_1 = \{\ (i,j) \mid (R'_1[i], R'_1[j])$ is a base pair in R_1, and $R'_2[i]$, $R'_2[j]$ are two (unpaired or paired) bases which do not form a base pair in $R_2\}$.

$PB_2 = \{\ (i,j) \mid (R'_2[i], R'_2[j])$ is a base pair in R_2, and $R'_1[i]$, $R'_1[j]$ are two (unpaired or paired) bases which do not form a base pair in $R_1\}$.

$PA_1 = \{\ (i,j) \mid (R'_1[i], R'_1[j])$ is a base pair in R_1, $R'_2[i]$ is a (unpaired or paired) base in R_2, and $R'_2[j] = '-'\}$.

$PA_2 = \{\ (i,j) \mid (R'_1[i], R'_1[j])$ is a base pair in R_1, $R'_2[j]$ is a (unpaired or paired) base in R_2, and $R'_2[i] = '-'\}$.

$PA_3 = \{\ (i,j) \mid (R'_2[i], R'_2[j])$ is a base pair in R_2, $R'_1[i]$ is a (unpaired or paired) base in R_1, and $R'_1[j] = '-'\}$.

$PA_4 = \{\ (i,j) \mid (R'_2[i], R'_2[j])$ is a base pair in R_2, $R'_1[j]$ is a (unpaired or paired) base in R_1, and $R'_1[i] = '-'\}$.

Under the linear gap penalty model, the cost of an alignment (S'_1, S'_2) is defined as follows.

$$\begin{aligned}
cost_{linear}((S'_1, S'_2)) = & \sum_{i \in SM} \Gamma(R'_1[i] \to R'_2[i]) \\
& + \sum_{i \in SD} \Gamma(R'_1[i] \to \lambda) \\
& + \sum_{i \in SI} \Gamma(\lambda \to R'_2[i]) \\
& + \sum_{(i,j) \in PM} \Gamma((R'_1[i], R'_1[j]) \to (R'_2[i], R'_2[j])) \\
& + \sum_{(i,j) \in PD} \Gamma((R'_1[i], R'_1[j]) \to \lambda) \\
& + \sum_{(i,j) \in PI} \Gamma(\lambda \to (R'_2[i], R'_2[j])) \\
& + \sum_{(i,j) \in PB_1} \Gamma_b((R'_1[i], R'_1[j])) \\
& + \sum_{(i,j) \in PB_2} \Gamma_b((R'_2[i], R'_2[j])) \\
& + \sum_{(i,j) \in PA_1} \Gamma((R'_1[i], R'_1[j]) \to (R'_2[i], \lambda)) \\
& + \sum_{(i,j) \in PA_2} \Gamma((R'_1[i], R'_1[j]) \to (\lambda, R'_2[j])) \\
& + \sum_{(i,j) \in PA_3} \Gamma((R'_1[i], \lambda) \to (R'_2[i], R'_2[j])) \\
& + \sum_{(i,j) \in PA_4} \Gamma((\lambda, R'_1[j]) \to (R'_2[i], R'_2[j])) \qquad (3.2)
\end{aligned}$$

Under the affine gap penalty model, we use *gap_cost* to represent the gap opening cost, and *#gap* to represent the number of gaps in (R'_1, R'_2). The cost of an alignment

(R'_1, R'_2) is defined as follows.

$$\begin{aligned} cost_{affine}((S'_1, S'_2)) = \quad & gap\_cost \times \#gap \\ & + cost_{linear}((S'_1, S'_2)) \end{aligned} \tag{3.3}$$

The cost of an alignment (R'_1, R'_2) can be determined in two steps. In the first step, we determine the operations performed on base pairs. In the second step, we determine the operations performed on unpaired bases and bases in the base pairs that have undergone base-pair bond breakings or base-pair alterings. For example, suppose that in the alignment, there exist $(i_1, j_1) \in P_1$ and $(i_2, j_2) \in P_2$, $R_1[i_1]$ and $R_2[i_2]$ are each aligned with a space $'-'$, and $R_1[j_1]$ is aligned with $R_2[j_2]$ but $R_1[j_1] \neq R_2[j_2]$. Then there are two base-pair altering operations and a base mismatch operation associated with them.

Given two RNA structure R_1 and R_2, our goal is to find the alignment with minimum cost:

$$Align(R_1, R_2) = \min_{(R'_1, R'_2)} \{cost((R'_1, R'_2))\}. \tag{3.4}$$

3.2 Hardness Results

In this section, we consider the problem of alignment between RNA structures where both structures are tertiary structures. In general, this problem is MAX SNP-hard.

When the edit operations base-pair bond breaking and base-pair altering are not allowed, there are several results from [21, 12, 19]. When all edit operations discussed in previous section are allowed, there is a result from [9].

When base-pair bond breaking and base-pair altering are not allowed, and the gap opening cost is zero, there is a result from [21].

Theorem 3.2.1 *The problem of computing the edit distance between two RNA tertiary structures is NP-hard.*

This means that this problem has no solutions in polynomial time unless P = NP. And there are two results from [12].

Theorem 3.2.2 *The problem of computing the edit distance between two RNA tertiary structures is MAX SNP-hard.*

This means that there is no polynomial time approximation scheme (PTAS) for this problem unless P = NP [15].

A maximization (similarity) version can also be considered, where the goal is to find a maximal-scoring edit sequence that can change one structure to the other. For the maximization version, the result is stronger than that for the minimization version.

Theorem 3.2.3 *For any $\delta < 1$, the maximization version of the problem of computing the edit distance between two RNA tertiary structures cannot be approximated within ratio $2^{\log^{\delta} n}$ in polynomial time unless $NP \in DTIME[2^{\text{poly} \log n}]$.*

When base-pair bond breaking and base-pair altering are not allowed, and the gap opening cost is greater than zero, there is a result from [19].

Theorem 3.2.4 *The problem of alignment between RNA structures with affine gap penalty where both structures are tertiary structures is MAX SNP-hard.*

When base-pair bond breaking and base-pair altering are allowed, and the gap opening cost is zero, there is a result from [9].

Theorem 3.2.5 *The problem of computing the edit distance between a RNA tertiary structure and a RNA primary structure is MAX SNP-hard.*

This result also implies that under this model, the problem of computing the edit distance between a RNA tertiary structure and a RNA secondary structure, and the problem of computing the edit distance between two RNA tertiary structures are MAX SNP-hard. These results can be extended to the model with affine gap penalty.

3.3 Wang and Zhang's Algorithm

In this section, we review Wang and Zhang's RNA alignment algorithm which serves as a basis of our algorithm.

Wang and Zhang presented an algorithm to compute the alignment between RNA structures for the case where aligned base pairs are non-crossing [19]. They treated a base pair as a unit and do not allow it to match to two unpaired bases. That is, base-pair bond breaking and base-pair altering operations are not allowed. Under this restricted assumption, when at least one of the RNA structures involved is a secondary structure, their algorithm can compute the optimal alignment in $O(stem(R_1) \times stem(R_2) \times |R_1| \times |R_2|)$ time. One more step, can be added to the algorithm to align tertiary base pairs. This step can be considered as a constrained alignment. Now we discuss their algorithm in detail.

Because base-pair bond breaking and base-pair altering are not allowed, we need to add two more conditions in defining the structural alignment:

- If $R'_1[i]$ is an unpaired base in R'_1, then either $R'_2[i]$ is an unpaired base in R'_2 or $R'_2[i] = '-'$. If $R'_2[i]$ is an unpaired base in R'_2, then either $R'_1[i]$ is an unpaired base in R'_1 or $R'_1[i] = '-'$.

- If $(R'_1[i], R'_1[j])$ is a base pair in R'_1, then either $(R'_2[i], R'_2[j])$ is a base pair in R'_2 or $R'_2[i] = R'_2[j] = '-'$. If $(R'_2[i], R'_2[j])$ is a base pair in R'_2, then either $(R'_1[i], R'_1[j])$ is a base pair in R'_1 or $R'_1[i] = R'_1[j] = '-'$.

Since aligning crossing base pairs is difficult (recall the hardness results in Section 3.2), we add one more condition in defining the structural alignment.

- If $(R'_1[i], R'_1[j])$ and $(R'_1[k], R'_1[l])$ are base pairs in R'_1 and $(R'_2[i], R'_2[j])$ and $(R'_2[k], R'_2[l])$ are base pairs in R'_2, then $(R'_1[i], R'_1[j])$ and $(R'_1[k], R'_1[l])$ are non-crossing in R'_1 and $(R'_2[i], R'_2[j])$ and $(R'_2[k], R'_2[l])$ are non-crossing in R'_2.

Therefore, even though the input RNA structures may have crossing base pairs, the aligned base pairs are non-crossing.

We use a bottom up dynamic programming algorithm to find the optimal alignment between R_1 and R_2. We consider the smaller substructures first and eventually consider the whole structures R_1 and R_2.

3.3.1 Property of Optimal Alignments

Consider two RNA structures R_1 and R_2, we use $\Gamma(\ )$ to define $\gamma(i,j)$ for $0 \leq i \leq |R_1|$ and $0 \leq j \leq |R_2|$.

If $i = p_{r_1}(i)$ and $j = p_{r_2}(j)$,

$$\gamma(i,0) = \Gamma(R_1[i] \rightarrow \lambda) \tag{3.5}$$

$$\gamma(0,j) = \Gamma(\lambda \rightarrow R_2[j]) \tag{3.6}$$

$$\gamma(i,j) = \Gamma(R_1[i] \rightarrow R_2[j]) \tag{3.7}$$

If $i' = p_{r_1}(i) < i$ and $j' = p_{r_2}(j) < j$,

$$\gamma(i',0) = \Gamma((R_1[i'], R_1[i]) \rightarrow \lambda)/2 \tag{3.8}$$

$$\gamma(i,0) = \Gamma((R_1[i'], R_1[i]) \rightarrow \lambda)/2 \tag{3.9}$$

$$\gamma(0,j') = \Gamma(\lambda \rightarrow (R_2[j'], R_2[j]))/2 \tag{3.10}$$

$$\gamma(0,j) = \Gamma(\lambda \rightarrow (R_2[j'], R_2[j]))/2 \tag{3.11}$$

$$\gamma(i,j) = \Gamma((R_1[i'], R_1[i]) \rightarrow (R_2[j'], R_2[j])) \tag{3.12}$$

From this definition, if $R_1[i]$ is a single base, then $\gamma(i,0)$ is the cost of deleting this base and if $R_1[i]$ is a base of a base pair, then $\gamma(i,0)$ is half of the cost of deleting this base pair. Therefore we distribute evenly the deletion cost of a base pair to its two bases. The meaning of $\gamma(0,j)$ is similar. When $i > 0$ and $j > 0$, $\gamma(i,j)$ is the cost of aligning base pairs $(R_1[i'], R_1[i])$ and $(R_2[j'], R_2[j])$.

We now consider the optimal alignment between $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$. We use $A(i_1, i_2; j_1, j_2)$ to represent the optimal alignment cost between $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$.

We use $D(i_1, i_2; j_1, j_2)$ to represent the optimal alignment cost such that $R_1[i_2]$ is aligned to $'-'$. We use $I(i_1, i_2; j_1, j_2)$ to represent the optimal alignment cost such that $R_2[j_2]$ is aligned to $'-'$.

In computing $A(i_1, i_2; j_1, j_2)$, $D(i_1, i_2; j_1, j_2)$ and $I(i_1, i_2; j_1, j_2)$, for any $i_1 \leq i \leq i_2$, if $p_{r_1}(i) < i_1$ or $i_2 < p_{r_1}(i)$, then $R_1[i]$ will be forced to be aligned to $'-'$; for any $j_1 \leq j \leq j_2$, if $p_{r_2}(j) < j_1$ or $j_2 < p_{r_2}(j)$, then $R_2[j]$ will be forced to be aligned to $'-'$. This is used to deal with two situations: aligning one base pair among crossing base pairs and deleting a base pair.

We can now consider how to compute the optimal alignment between $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$.

Lemma 3.3.1

$$A(\emptyset; \emptyset) = 0 \tag{3.13}$$

$$D(\emptyset; \emptyset) = \mathit{gap\_cost} \tag{3.14}$$

$$I(\emptyset; \emptyset) = \mathit{gap\_cost} \tag{3.15}$$

Proof: For $A(\emptyset; \emptyset)$, consider $A(i_1, i_1; j_1, j_1)$. If the optimal alignment results from aligning $R_1[i_1]$ to $R_2[j_1]$, then we only need to account for the cost for aligning $R_1[i_1]$ to $R_2[j_1]$. Hence we may set $A(\emptyset; \emptyset) = 0$.

For $D(\emptyset; \emptyset)$, consider $D(i_1, i_1; \emptyset)$ by which $R_1[i_1]$ is aligned to $'-'$. Aligning $R_1[i_1]$ to $'-'$ opens a gap, so we need to charge gap opening penalty for it. Hence we may set $D(\emptyset; \emptyset) = \mathit{gap\_cost}$.

Similarly, we can set $I(\emptyset; \emptyset) = \mathit{gap\_cost}$. □

Lemma 3.3.2 *For* $i > 0$,

$$D(i_1, i; \emptyset) = D(i_1, i-1; \emptyset) + \gamma(i, 0) \tag{3.16}$$

$$A(i_1, i; \emptyset) = D(i_1, i; \emptyset) \tag{3.17}$$

$$I(i_1, i; \emptyset) = D(i_1, i; \emptyset) + \mathit{gap\_cost} \tag{3.18}$$

For $j > 0$,

$$I(\emptyset; j_1, j) = I(\emptyset; j_1, j-1) + \gamma(0, j) \tag{3.19}$$

$$A(\emptyset; j_1, j) = I(\emptyset; j_1, j) \tag{3.20}$$

$$D(\emptyset; j_1, j) = I(\emptyset; j_1, j) + gap\_cost \tag{3.21}$$

Proof: For $D(i_1, i; \emptyset)$, by definition $R_1[i]$ is aligned to $'-'$, hence we have the $\gamma(i, 0)$ term, and $R_1[i_1, i-1]$ is aligned to $\emptyset$. That is, each element in $R_1[i_1, i-1]$ is aligned to $'-'$, by which we know that $R_1[i-1]$ is aligned to $'-'$. Hence we have $D(i_1, i; \emptyset) = D(i_1, i-1; \emptyset) + \gamma(i, 0)$.

For $A(i_1, i; \emptyset)$, this is the optimal alignment between $R_1[i_1, i]$ and $\emptyset$. Thus each element in $R_1[i_1, i]$ is aligned to $'-'$, by which we know that $R_1[i]$ is aligned to $'-'$. Hence we have $A(i_1, i; \emptyset) = D(i_1, i; \emptyset)$.

For $I(i_1, i; \emptyset)$, consider $I(i_1, i; j_1, j_1)$, the optimal alignment between $R_1[i_1, i]$ and $R_2[j_1, j_1]$ that ends with $R_2[j_1]$ aligned to $'-'$. Thus $R_1[i_1, i]$ is aligned to $\emptyset$. That is, each element in $R_1[i_1, i]$ is aligned to $'-'$, by which we know that $R_1[i]$ is aligned to $'-'$. Aligning $R_2[j_1]$ to $'-'$ opens a gap, so we need to charge gap opening penalty for it. Hence we have $I(i_1, i; \emptyset) = D(i_1, i; \emptyset) + gap\_cost$.

Similarly, we can obtain other three formulas. □

Lemma 3.3.3 *For* $i_1 \le i \le i_2$ *and* $j_1 \le j \le j_2$,

$$D(i_1, i; j_1, j) = \min \begin{cases} D(i_1, i-1; j_1, j) + \gamma(i, 0) \\ A(i_1, i-1; j_1, j) + \gamma(i, 0) + gap\_cost \end{cases} \tag{3.22}$$

Proof: We use $M(i_1, i; j_1, j)$ to represent the optimal alignment cost such that $R_1[i]$

is aligned to $R_2[j]$. Then

$$A(i_1, i; j_1, j) = \min \begin{cases} D(i_1, i; j_1, j) \\ I(i_1, i; j_1, j) \\ M(i_1, i; j_1, j) \end{cases}$$

According to the definition of $D(i_1, i; j_1, j)$, $R_1[i]$ is aligned to $'-'$, hence $\gamma(i, 0)$. We consider $R_1[i-1]$ and $R_2[j]$, there are exactly the following cases.

(1) $R_1[i-1]$ is aligned to $'-'$. $D(i_1, i; j_1, j)$ is from $D(i_1, i-1; j_1, j)$, then aligning $R_1[i]$ to $'-'$ does not open a gap. Therefore there is no gap opening penalty.

(2) $R_2[j]$ is aligned to $'-'$. $D(i_1, i; j_1, j)$ is from $I(i_1, i-1; j_1, j)$, then aligning $R_1[i]$ to $'-'$ opens a gap. Therefore there is a gap opening penalty.

(3) $R_1[i-1]$ is aligned to $R_2[j]$. $D(i_1, i; j_1, j)$ is from $M(i_1, i-1; j_1, j)$, then aligning $R_1[i]$ to $'-'$ opens a gap. Therefore there is a gap opening penalty.

So we have the following recursion.

$$\begin{aligned} D(i_1, i; j_1, j) &= \min \begin{cases} D(i_1, i-1; j_1, j) + \gamma(i, 0) \\ I(i_1, i-1; j_1, j) + \gamma(i, 0) + \mathit{gap\_cost} \\ M(i_1, i-1; j_1, j) + \gamma(i, 0) + \mathit{gap\_cost} \end{cases} \\ &= \min \begin{cases} D(i_1, i-1; j_1, j) + \gamma(i, 0) \\ D(i_1, i-1; j_1, j) + \gamma(i, 0) + \mathit{gap\_cost} \\ I(i_1, i-1; j_1, j) + \gamma(i, 0) + \mathit{gap\_cost} \\ M(i_1, i-1; j_1, j) + \gamma(i, 0) + \mathit{gap\_cost} \end{cases} \\ &= \min \begin{cases} D(i_1, i-1; j_1, j) + \gamma(i, 0) \\ A(i_1, i-1; j_1, j) + \gamma(i, 0) + \mathit{gap\_cost} \end{cases} \end{aligned}$$

$\square$

Lemma 3.3.4 *For $i_1 \le i \le i_2$ and $j_1 \le j \le j_2$,*

$$I(i_1, i; j_1, j) = \min \begin{cases} I(i_1, i; j_1, j-1) + \gamma(0, j) \\ A(i_1, i; j_1, j-1) + \gamma(0, j) + gap\_cost \end{cases} \tag{3.23}$$

Proof: Similar to Lemma 3.3.3. □

Lemma 3.3.5 *For $i_1 \le i \le i_2$ and $j_1 \le j \le j_2$, if $i = p_{r_1}(i)$ and $j = p_{r_2}(j)$, then*

$$A(i_1, i; j_1, j) = \min \begin{cases} D(i_1, i; j_1, j) \\ I(i_1, i; j_1, j) \\ A(i_1, i-1; j_1, j-1) + \gamma(i, j) \end{cases} \tag{3.24}$$

if $i_1 \le p_{r_1}(i) < i$ and $j_1 \le p_{r_2}(j) < j$, then

$$A(i_1, i; j_1, j) = \min \begin{cases} D(i_1, i; j_1, j) \\ I(i_1, i; j_1, j) \\ A(i_1, p_{r_1}(i) - 1; j_1, p_{r_2}(j) - 1) \\ +A(p_{r_1}(i) + 1, i-1; p_{r_2}(j) + 1, j-1) + \gamma(i, j) \end{cases} \tag{3.25}$$

otherwise,

$$A(i_1, i; j_1, j) = \min \begin{cases} D(i_1, i; j_1, j) \\ I(i_1, i; j_1, j) \end{cases} \tag{3.26}$$

Proof: Consider the optimal alignment between $R_1[i_1, i]$ and $R_2[j_1, j]$. There are three cases: 1. $i = p_{r_1}(i)$ and $j = p_{r_2}(j)$, 2. $i_1 \le p_{r_1}(i) < i$ and $j_1 \le p_{r_2}(j) < j$, and 3. all the other cases.

For case 1, since $i = p_{r_1}(i)$ and $j = p_{r_2}(j)$, both $R_1[i]$ and $R_2[j]$ are unpaired bases. In the optimal alignment, $R_1[i]$ may be aligned to $'-'$, $R_2[j]$ may be aligned to $'-'$, or $R_1[i]$ may be aligned to $R_2[j]$. Therefore we take the minimum of the three cases.

For case 2, since $i_1 \leq p_{r_1}(i) < i$ and $j_1 \leq p_{r_2}(j) < j$, both $(R_1[p_{r_1}(i)], R_1[i])$ and $(R_2[p_{r_2}(j)], R_2[j])$ are base pairs. In the optimal alignment, $(R_1[p_{r_1}(i)], R_1[i])$ may be aligned to $('-', '-')$, $(R_2[p_{r_2}(j)], R_2[j])$ may be aligned to $('-', '-')$, or $(R_1[p_{r_1}(i)], R_1[i])$ may be aligned to $(R_2[p_{r_2}(j)], R_2[j])$.

If $(R_1[p_{r_1}(i)], R_1[i])$ is aligned to $('-', '-')$, then $A(i_1, i; j_1, j) = D(i_1, i; j_1, j)$. If $(R_2[p_{r_2}(j)], R_2[j])$ is aligned to $('-', '-')$ then $A(i_1, i; j_1, j) = I(i_1, i; j_1, j)$.

If $(R_1[p_{r_1}(i)], R_1[i])$ is aligned to $(R_2[p_{r_2}(j)], R_2[j])$, then the optimal alignment between $R_1[i_1, i]$ and $R_2[j_1, j]$ is partitioned into three parts: 1. the optimal alignment between $R_1[i_1, p_{r_1}(i)-1]$ and $R_2[j_1, p_{r_2}(j)-1]$, 2. the optimal alignment between $R_1[p_{r_1}(i)+1, i-1]$ and $R_2[p_{r_2}(j)+1, j-1]$, and 3. the alignment of $(R_1[p_{r_1}(i)], R_1[i])$ to $(R_2[p_{r_2}(j)], R_2[j])$. Hence we have $A(i_1, i; j_1, j) = A(i_1, p_{r_1}(i) - 1; j_1, p_{r_2}(j) - 1) + A(p_{r_1}(i) + 1, i - 1; p_{r_2}(j) + 1, j - 1) + \gamma(i, j)$. Note that any base pair across $(R_1[p_{r_1}(i)], R_1[i])$ or $(R_2[p_{r_2}(j)], R_2[j])$ should be aligned to $'-'$ and the cost of such an alignment has already been included in part 1 and part 2.

In case 3, we consider all the other possibilities in which we cannot align $R_1[i]$ to $R_2[j]$. This includes many cases.

- sub-case 1: $R_1[i]$ is a single base and $R_2[j]$ is a base in a base pair. This means that we have to align $R_1[i]$ to $'-'$ or align $R_2[j]$ to $'-'$.

- sub-case 2: $R_1[i]$ is a base in a base pair and $R_2[j]$ is a single base. This is similar to sub-case 1.

- sub-case 3: $R_1[i]$ is a base in a base pair and $p_{r_1}(i) > i$. This means that $R_1[p_{r_1}(i)]$ is outside the interval $[i_1, i]$ and we have to align $R_1[i]$ to $'-'$.

- sub-case 4: $R_2[j]$ is a base in a base pair and $p_{r_2}(j) > j$. This is similar to sub-case 3. Together with sub-case 3, this implies that when $p_{r_1}(i) > i$ and $p_{r_2}(j) > j$, even if $R_1[i] = R_2[j]$, we cannot align them to each other.

- sub-case 5: $R_1[i]$ is a base in a base pair and $p_{r_1}(i) < i_1$. This is similar to sub-case 3. Together with sub-case 3, we know that if a base pair is across an aligned base pair, then it has to be aligned to '$-$'.

- sub-case 6: $R_2[j]$ is a base in a base pair and $p_{r_2}(j) < j_1$. This is similar to sub-case 5.

$\square$

3.3.2 Algorithm

From the above lemmas, we can compute $Align(R_1, R_2) = A(1, |R_1|; 1, |R_2|)$ using a bottom up approach. From Lemma 3.3.5, we only need to compute those $A(i_1, i_2; j_1, j_2)$ such that $(R_1[i_1 - 1], R_1[i_2 + 1])$ is a base pair in R_1 and $(R_2[j_1 - 1], R_2[j_2 + 1])$ is a base pair in R_2.

Given R_1 and R_2, we can first compute sorted base pair lists L_1 for R_1 and L_2 for R_2. For each pair of base pairs $L_1[i] = (i_1, i_2)$ and $L_2[j] = (j_1, j_2)$, we use Lemma 3.3.1 through Lemma 3.3.5 to compute $A(i_1 + 1, i_2 - 1; j_1 + 1, j_2 - 1)$.

Let R_1 and R_2 be the two given RNA structures and P_1 and P_2 be the number of base pairs in R_1 and R_2 respectively. The time to compute $A(i_1, i_2; j_1, j_2)$ is $O((i_2 - i_1)(j_2 - j_1))$ which is bounded by $O(|R_1| \times |R_2|)$. The time complexity of the algorithm in worst case is $O(P_1 \times P_2 \times |R_1| \times |R_2|)$. We can improve our algorithm so that the worst case running time is $O(stem(R_1) \times stem(R_2) \times |R_1| \times |R_2|)$ where $stem(R_1)$ and $stem(R_2)$ are the number of stems, in R_1 and R_2 respectively. The space complexity of the algorithm is $O(|R_1| \times |R_2|)$.

Notice that when one of the RNAs is a secondary structure, this algorithm computes the optimal solution of the problem. Also, since the number of tertiary base pairs is relatively small compared with the number of secondary base pairs, we can use this algorithm to compute the alignment between RNA tertiary structures. Essentially the algorithm tries to find the best sets of non-crossing base pairs to align and delete tertiary interactions. Although this is not an optimal solution, in practice it would produce a reasonable result by aligning most of the base pairs.

3.3.3 Constrained Alignment

Wang and Zhang also proposed a heuristic method to align tertiary structures in [19]. The method is as follows.

Given two RNA tertiary structures, we first apply the alignment algorithm presented in Section 3.3.2 to produce an alignment where aligned base pairs are non-crossing and then, using these aligned base pairs as the constraints, we align tertiary base pairs if they are compatible with the base pairs already aligned. The second step can be considered as a *constrained alignment* problem where the goal is to find the optimal alignment using these aligned base pairs as the constraints.

3.4 Möhl *et al.*'s Algorithm

In this section, we review Möhl *et al.*'s RNA alignment algorithm which serves as a basis of our algorithm.

Möhl *et al.* presented a fixed parameter tractable algorithm to compute the optimal alignment between two RNA tertiary structures in [13]. The parameter, which determines the exponential runtime, depends on how complex the crossing stems are arranged. They used w_m to denote base mismatch cost, w_d to denote base insertion/deletion cost, w_{am} to denote base-pair mismatch cost (it costs $w_{am}/2$ if one base in the base pair is replaced or w_{am} if both bases in the base pair are replaced), w_r to denote base-pair insertion/deletion cost, w_b to denote base-pair bond breaking cost, and w_a to denote base-pair altering cost. The algorithm is under the restricted score schemes $w_a = (w_b + w_r)/2$. This algorithm is a generalization of the algorithm in [9] to tertiary structures.

The main idea of the algorithm is partitioning the set of arc pairs $P_1 \times P_2$ into a set

NC of "non-crossing" arc pairs and a set of "crossing" arc pairs $CR = P_1 \times P_2 - NC$ such that the algorithm can apply a polynomial alignment method for the arc pairs in NC and an exponential alignment method for the arc pairs in CR.

3.4.1 Partition Arc Pairs into Crossing Arc Pairs and Non-crossing Arc Pairs

Now we discuss how to do the partition of the set of arc pairs.

Möhl *et al.* give a definition about valid partition of the set of arc pairs $P_1 \times P_2$ into NC and CR in [13].

Definition 3.4.1 *A partition of $P_1 \times P_2$ into NC and CR is valid if and only if for all $a, a' \in NC$ it holds that a and a' do not cross.*

And they give a partition method according to left crossing arcs.

Lemma 3.4.2 *The partition of $P_1 \times P_2$ into $CR = CR_1 \times CR_2 = \{p_1 \in P_1 \mid p_1$ is left crossing$\} \times \{p_2 \in P_2 \mid p_2$ is left crossing$\}$ and $NC = P_1 \times P_2 - CR$ is valid.*

It is easy to see Lemma 3.4.2 holds, because for two arbitrary crossing arc pairs, one of them is in CR. Thus a valid partition can be obtained if CR_1 and CR_2 contain all left crossing arcs. Analogously, a valid partition can be obtained if CR_1 and CR_2 contain all right crossing arcs.

Lemma 3.4.3 *The partition of $P_1 \times P_2$ into $CR = CR_1 \times CR_2 = \{p_1 \in P_1 \mid p_1$ is right crossing$\} \times \{p_2 \in P_2 \mid p_2$ is right crossing$\}$ and $NC = P_1 \times P_2 - CR$ is valid.*

In the example shown in Figure 3.5 in Section 3.1, $P_1 = \{1, 2, 3\}$ and $P_2 = \{$I, II, III$\}$. Applying Lemma 3.4.2 to the example, we can get $CR = \{2, 3\} \times \{$II, III$\}$ $= \{(2,$ II$), (2,$ III$), (3,$ II$), (3,$ III$)\}$ and $NC = \{(1,$ I$), (1,$ II$), (1,$ III$), (2,$ I$), (3,$ I$)\}$. Applying Lemma 3.4.3 to the example, we can get $CR = \{1, 2\} \times \{$I, II$\} = \{(1,$ I$),$ $(1,$ II$), (2,$ I$), (2,$ II$)\}$ and $NC = \{(1,$ III$), (2,$ III$), (3,$ I$), (3,$ II$), (3,$ III$)\}$.

Because the algorithm applies a polynomial alignment method for the arc pairs in NC and an exponential alignment method for the arc pairs in CR, i.e. the algorithm handles arc pairs in NC more efficiently than arc pairs in CR, we want to make the cardinality of CR as small as we can. A good partition should be local minimal, i.e. it becomes invalid if any element is removed from CR. Unfortunately, the partition according to Lemma 3.4.2 or Lemma 3.4.3 may not be local minimal (for the example shown in Figure 3.5 in Section 3.1, the partition according to Lemma 3.4.2 puts arc pair (3, III) in CR which actually can be moved to NC; similar for the partition according to Lemma 3.4.3). We will propose a method to optimize the partition result in Section 4.2. The optimized partition will be local minimal.

3.4.2 Precomputation of Stem Pairs

Because the algorithm applies an exponential alignment method for the arc pairs in CR and that will cost a lot of runtime, we should consider aligning whole crossing stems in one step. In order to align whole stems in one step, we need to group arc pairs in CR into pairs of stems.

Before we discuss how to do the precomputation, we need to introduce some notions given in [13].

Möhl *et al.* give a definition of stem which is different from the original one introduced in Section 1.2. We call the stem defined by the new definition "extended stem". In the remaining part of this thesis, we will use this definition. We simply refer to extended stem as "stem", and the stem defined by the original definition as "traditional stem".

A *stem* Q in P (for $P \in \{P_1, P_2\}$) is defined as a set of arcs $\{p_1, \cdots, p_k\} \subseteq P$ with $p_1^L < \cdots < p_k^L < p_k^R < \cdots < p_1^R$ such that no end of arcs in $P - Q$ is in one of the intervals $[p_1^L..p_k^L]$ or $[p_k^R..p_1^R]$. Notice that stems do not need to be maximal, and can include bulges and internal loops (that is, there can be bases between two adjacent arcs in the same stem) according to this definition. An example of stem is shown in

Figure 3.7.

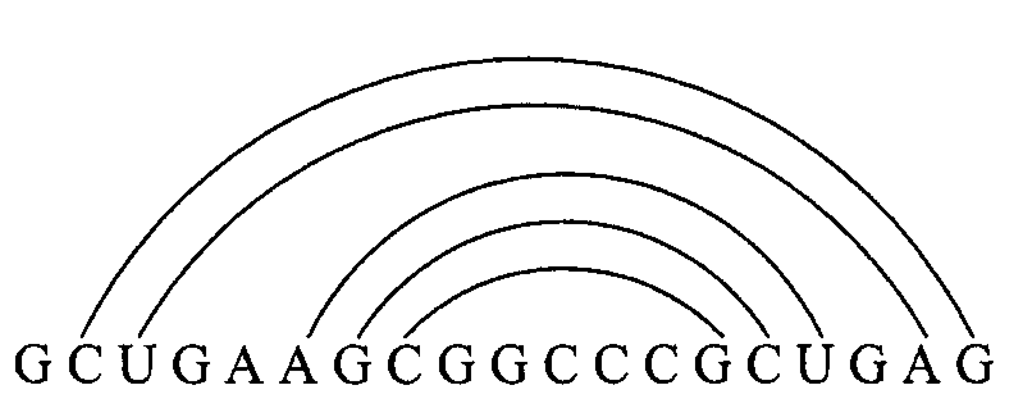


Figure 3.7: An example of extend stem

The *stem pair* of two stems $Q_1 \subseteq P_1$ and $Q_2 \subseteq P_2$ is characterized by the pair (a_O, a_I) of arc pairs, where $a_O = (p_{O_1}, p_{O_2})$ is the pair of the outermost arcs and $a_I = (p_{I_1}, p_{I_2})$ is the pair of the innermost arcs of Q_1 and Q_2, i.e. Q_k consists of the arcs $P_k \cap [p^L_{O_k}..p^L_{I_k}] \times [p^R_{I_k}..p^R_{O_k}]$ $(k = 1, 2)$. The stem pair *covers* an arc pair a iff $a \in Q_1 \times Q_2$. A stem pair is *realized in an alignment* (R'_1, R'_2) iff a_O and a_I are realized in (R'_1, R'_2). According to this definition, we know that only the outermost arcs and the innermost arcs are required to be matched. Figure 3.8 gives an illustration of a stem pair $(a_O, a_I) = ((p_{O_1}, p_{O_2}), (p_{I_1}, p_{I_2}))$ which covers the dotted arc pair (p_1, p_2).

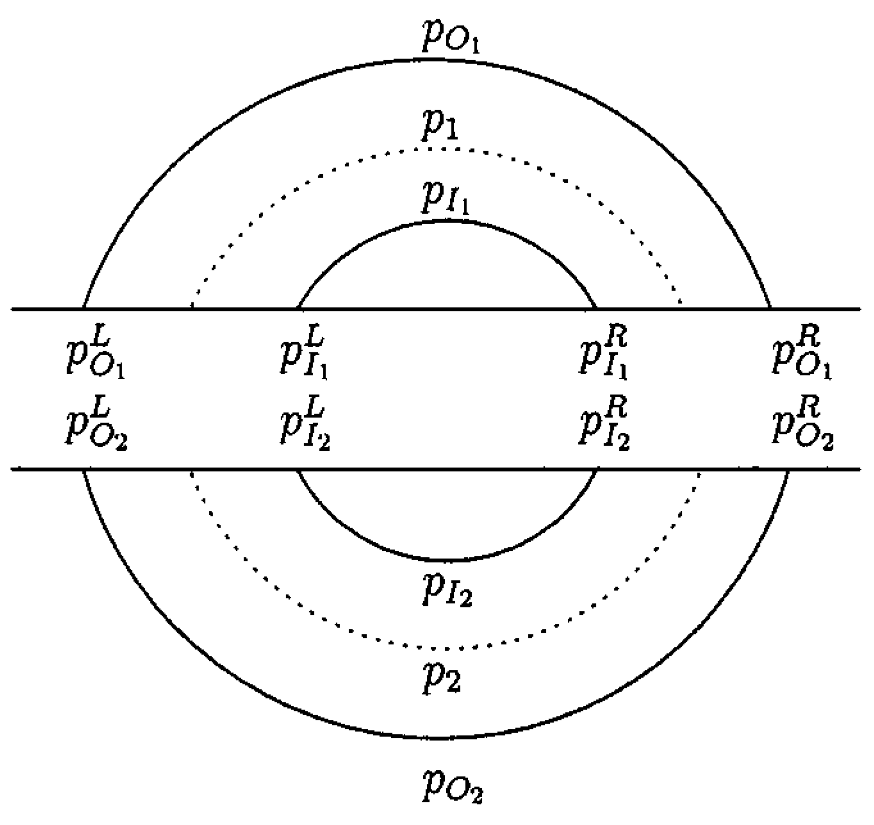


Figure 3.8: An example of stem pair

The set of all stem pairs (a_O, a_I) where $\{a_O, a_I\} \subseteq CR$ is denoted as ST_{CR}. A stem pair (a_O, a_I) is *open for a subalignment* $(R'_1, R'_2)[i_1, i_2; j_1, i_2]$ *in an alignment* (R'_1, R'_2) iff a_O, a_I are open for $(R'_1, R'_2)[i_1, i_2; j_1, i_2]$. The *set of maximal open stem pairs of a subalignment* $(R'_1, R'_2)[i_1, i_2; j_1, i_2]$ *in an alignment* (R'_1, R'_2) is the smallest set M of open stem pairs of $(R'_1, R'_2)[i_1, i_2; j_1, i_2]$ such that each $a \in OA((R'_1, R'_2)[i_1, i_2; j_1, i_2])$ is covered by a stem pair in M.

Now we discuss how to do the precomputation.

We precompute the cost to align each stem pair $(a_O, a_I) \in ST_{CR}$ as the value of an item $S(a_O, a_I)$. The cost of aligning a stem pair is the cost to align bases in the stem. Formally, for $a_O = (p_{O_1}, p_{O_2})$ and $a_I = (p_{I_1}, p_{I_2})$, the value of $S(a_O, a_I)$ is the cost to align $R_1[p^L_{O_1}, p^L_{I_1}]$ to $R_2[p^L_{O_2}, p^L_{I_2}]$ and simultaneously $R_1[p^R_{I_1}, p^R_{O_1}]$ to $R_2[p^R_{I_2}, p^R_{O_2}]$.

The computation of S items is based on temporary items $S'(i, i'; j, j'; a_I)$ that correspond to $S(((i, i'), (j, j')); a_I)$ if $((i, i'), (j, j'))$ is an arc pair, but are not limited to this case. $S'(i, i'; j, j'; ((i_a, i'_a), (j_a, j'_a)))$ is invalid if $i > i_a$, $i' < i'_a$, $j > j_a$ or $j' < j'_a$. The recursion to compute S' items given in [13] is as follows.

$$S'(i, i'; j, j'; a_I) =$$
$$\min \begin{cases} S'(i+1, i'; j, j'; a_I) + w_d + \psi_1(i)(w_r/2 - w_d) & (1) \\ S'(i, i'; j+1, j'; a_I) + w_d + \psi_2(j)(w_r/2 - w_d) & (2) \\ S'(i, i'-1; j, j'; a_I) + w_d + \psi_1(i')(w_r/2 - w_d) & (3) \\ S'(i, i'; j, j'-1; a_I) + w_d + \psi_2(j')(w_r/2 - w_d) & (4) \\ S'(i+1, i'; j+1, j'; a_I) + \chi(i, j)w_m + (\psi_1(i) + \psi_2(j))w_b/2 & (5) \\ S'(i, i'-1; j, j'-1; a_I) + \chi(i', j')w_m + (\psi_1(i') + \psi_2(j'))w_b/2 & (6) \\ \text{if } ((i, i'), (j, j')) \in CR & \\ \quad S'(i+1, i'-1; j+1, j'-1; a_I) + (\chi(i, j) + \chi(i', j'))w_{am}/2 & (7) \end{cases} \tag{3.27}$$

The functions χ and $\psi_k (k = 1, 2)$ in previous recursion are defined as follows.

$$\chi(i,j) = \begin{cases} 1 & \text{if } R_1[i] \neq R_2[j] \\ 0 & \text{otherwise} \end{cases} \tag{3.28}$$

$$\psi_k(i) = \begin{cases} 1 & \text{if } \exists j : (i,j) \in P_k \text{ or } (j,i) \in P_k \text{ (for } k = 1,2) \\ 0 & \text{otherwise} \end{cases} \tag{3.29}$$

We consider positions i, i', j and j', there are exactly the following cases directly corresponding to the recursion shown in Eq. 3.27.

(1) $R_1[i]$ is aligned to $'-'$. If $R_1[i]$ is a single base, this is due to a base deletion with cost w_d. Otherwise, the base pair which $R_1[i]$ is involved in is either deleted or altered, with cost $w_r/2$.

(2) $R_2[j]$ is aligned to $'-'$. This is analogous to case (1).

(3) $R_1[i']$ is aligned to $'-'$. This is analogous to case (1).

(4) $R_2[j']$ is aligned to $'-'$. This is analogous to case (1).

(5) $R_1[i]$ and $R_2[j]$ are matched by the alignment, but no arc pair involving $(R_1[i], R_2[j])$ is realized. Thus all adjacent arcs are broken, each resulting in cost $w_b/2$. If $R_1[i]$ and $R_2[j]$ mismatch, this causes an additional cost w_m.

(6) $R_1[i']$ and $R_2[j']$ are matched by the alignment, but no arc pair involving $(R_1[i'], R_2[j'])$ is realized. This is analogous to case (5).

(7) $((i,i'),(j,j')) \in CR$ and this arc pair is realized. If $R_1[i]$ and $R_2[j]$ mismatch, this causes a cost $w_{am}/2$. If $R_1[i']$ and $R_2[j']$ mismatch, this causes an additional cost $w_{am}/2$.

The alignment of the innermost arc pair is computed as $S(i,i';j,j';((i,i'),(j,j'))) = (\chi(i,j) + \chi(i,j))w_{am}/2$ and step by step enlarged with the recursion given in Eq. 3.27, where implicitly cases relying on invalid items are skipped. By the recursion for S', only the arc pair a_I is guaranteed to be realized in the precomputed optimal stem alignments. But we want to consider only items

$S(a_O, a_I)$ where both a_I and a_O are realized in the core algorithm discussed in 3.4.4 in order to avoid ambiguity in the recursion. Thus items where a_O is not realized are defined as invalid, and cases referring to these items are skipped in the core algorithm.

In the original paper [13], it is not clear when the computation of the S' items will terminate. We assume that for each S' item, the computation will terminate when the outermost arc pair of the maximal stem pair which covers a_I is encountered. We will add a new concept called "local maximal stem pair" in Section 4.1, and show that we only need to compute the alignment of local maximal stem pairs in Section 4.5.2.

3.4.3 Property of Optimal Alignment

We consider the optimal alignment between $R_1[i, i']$ and $R_2[j, j']$. We use $D(i, i'; j, j'|M)$ to represent the optimal alignment cost between $R_1[i, i']$ and $R_2[j, j']$ where $M \subseteq ST_{CR}$ is its set of maximal open stem pairs. A helpful intuition of M in the D items is that one end of each stem pair in M is aligned inside the range $(i, i'; j, j')$ and the other end is required to be aligned outside the range $(i, i'; j, j')$.

$D(i, i'; j, j'|M)$ is valid if $i' \geq i - 1$, $j' \geq j - 1$, and there is an alignment (R_1', R_2') such that M is the set of maximal open stem pairs of the subalignment $(R_1', R_2')[i_1, i_2; j_1, i_2]$ in (R_1', R_2').

We can now consider how to compute the optimal alignment between $R_1[i, i']$ and $R_2[j, j']$.

Assume an optimal alignment *Align* with a subalignment *Align*$[i, i'; j, j']$. We consider positions i' and j', there are exactly the following cases directly corresponding to the recursion shown in Eq. 3.30. An intuitive illustration is given in Figure 3.9[1]. The red dotted arcs in Figure 3.9 represent the set of open stem pairs M.

[1]Image quoted from [13].

$$
\begin{aligned}
&D(i,i';j,j'|M) = \\
&\min \begin{cases}
D(i,i'-1;j,j'|M) + w_d + \psi_1(i')(w_r/2 - w_d) & (1) \\
D(i,i';j,j'-1|M) + w_d + \psi_2(j')(w_r/2 - w_d) & (2) \\
D(i,i'-1;j,j'-1|M) + \chi(i',j')w_m + (\psi_1(i') + \psi_2(j'))w_b/2 & (3) \\
\text{if there exist some } i_1,\, j_1 \text{ with } ((i_1,i'),(j_1,j')) \in NC & \\
\quad \left\{ \min \left\{ \begin{array}{c} D(i,i_1-1;j,j_1-1|M_1) \\ +D(i_1+1,i'-1;j_1+1,j'-1|M_2) \\ +(\chi(i_1,j_1)+\chi(i',j'))w_{am}/2 \end{array} \right. \left| \begin{array}{l} M_1, M_2 \subseteq ST_{CR}, \text{ where} \\ M = (M_1 \cup M_2) - (M_1 \cap M_2) \end{array} \right. \right\} & (4) \\
\text{if there exists some } (a_O,a_I) \in M \text{ with} & \\
\quad \nwarrow a_O = (i_1,j_1) \wedge \nwarrow a_I = (i',j') \text{ or } \searrow a_I = (i_1,j_1) \wedge \searrow a_O = (i',j') & \\
\quad D(i,i_1-1;j,j_1-1|M - \{(a_O,a_I)\}) + S(a_O,a_I)/2 & (5) \\
\left\{ \min \left\{ \begin{array}{c} D(i,i_1-1;j,j_1-1|M \cup \{(a_O,a_I)\}) \\ +S(a_O,a_I)/2 \end{array} \right. \left| \begin{array}{c} (a_O,a_I) \in ST_{CR}, \text{ where} \\ \searrow a_O = (i',j') \text{ and} \\ \searrow a_I = (i_1,j_1) \end{array} \right. \right\} & (6)
\end{cases}
\end{aligned}
\tag{3.30}
$$

(1) $R_1[i']$ is aligned to $'-'$. If $R_1[i']$ is a single base, this is due to a base deletion with cost w_d. Otherwise, the base pair which $R_1[i']$ is involved in is either deleted or altered, with cost $w_r/2$.

(2) $R_2[j']$ is aligned to $'-'$. This is analogous to case (1).

(3) $R_1[i']$ and $R_2[j']$ are matched by the alignment, but no arc pair involving $(R_1[i'], R_2[j'])$ is realized. Thus all adjacent arcs are broken, each resulting in cost $w_b/2$. If $R_1[i']$ and $R_2[j']$ mismatch, this causes an additional cost w_m.

(4) $R_1[i']$ and $R_2[j']$ are matched by the alignment, and an arc pair $((i_1,i'),(j_1,j'))$ with right end (i',j') is realized. Then the subalignment is partitioned into three parts:

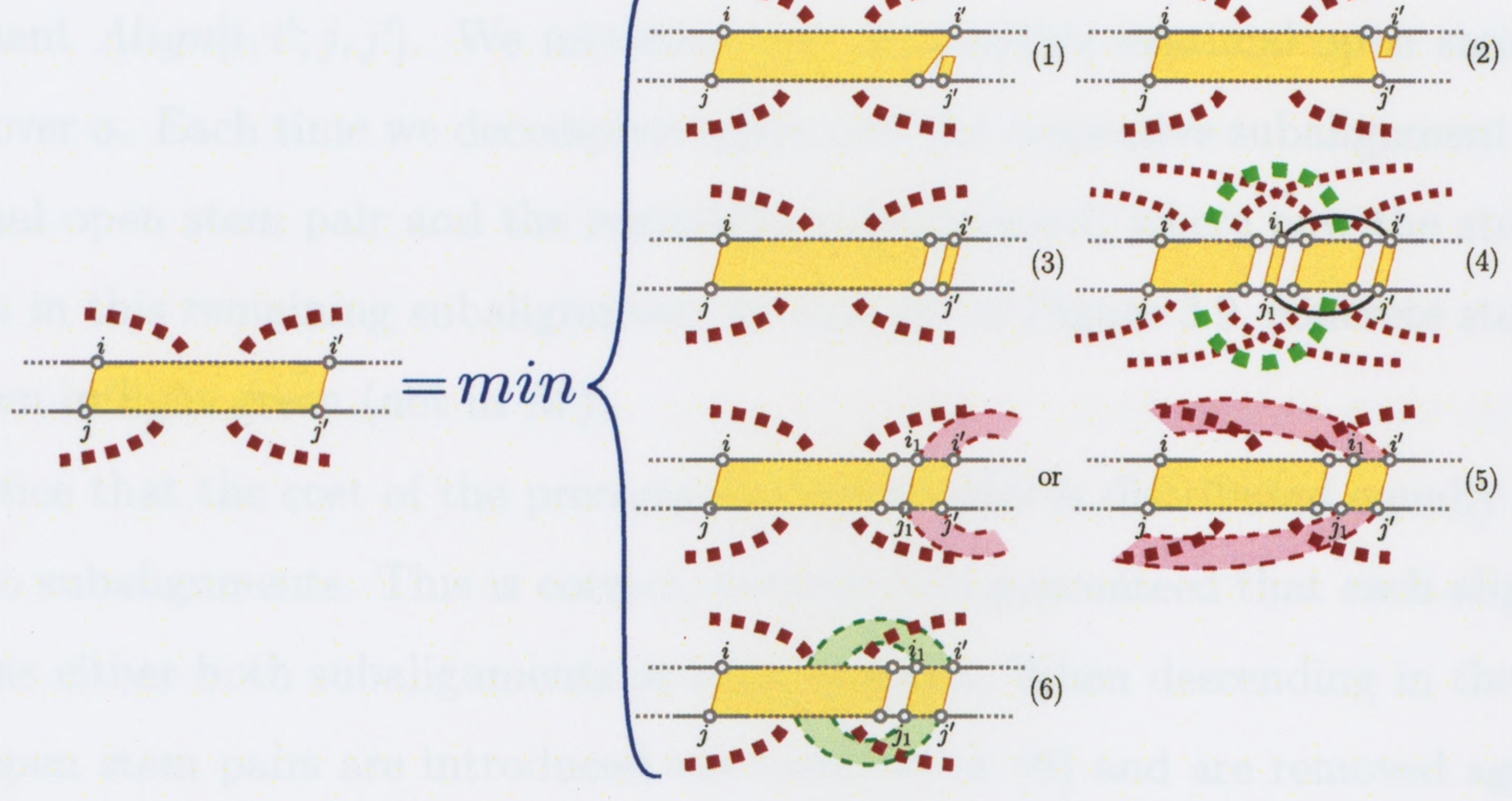


Figure 3.9: Illustration of Möhl *et al.*'s recursion to compute D items

the arc pair, the subalignment before the arc pair and the subalignment inside the arc pair. The cost of aligning the arc pair $((i_1, i'), (j_1, j'))$ is 0 (if both pairs $R_1[i_1]-R_2[j_1]$, $R_1[i_1] - R_2[j_1]$ match), or $w_{am}/2$ (if one of the pairs $R_1[i_1] - R_2[j_1]$, $R_1[i_1] - R_2[j_1]$ mismatch), or w_{am} (if both pairs $R_1[i_1] - R_2[j_1]$, $R_1[i_1] - R_2[j_1]$ mismatch). For the two subalignments, their corresponding maximal open stem pairs sets are denoted by M_1 and M_2, respectively. We need to minimize over all possible alternatives. Particularly, these include the cases where M_1 and M_2 contain open stem pairs which are not contained in M, that is, those cases where $M_1 \cap M_2 \neq \emptyset$. In case (4) of Figure 3.9, the green dotted arcs represent the set of stem pairs shared between the two alignment fragments (i.e. $M_1 \cap M_2$) and the red dotted arcs represent the remaining elements of $M_1 \cup M_2$, which make up M.

(5) An arc pair a in CR has left or right end in (i', j') and a is open for the subalignment $Align[i, i'; j, j']$. The maximal open stem pair that cover a is contained in M and therefore uniquely determined. Hence we can decompose into the respective subalignment of this maximal open stem pair and the remaining subalignment, where this stem pair is no more open. In case (5) of Figure 3.9, concrete stem pair is shown in light red (in M).

(6) An arc pair a in CR has right end in (i', j') and a is not open for the sub-

alignment $Align[i, i'; j, j']$. We minimize over all possible maximal open stem pairs that cover a. Each time we decompose again into the respective subalignment of this maximal open stem pair and the remaining subalignment, where now the stem pair is open in this remaining subalignment. In case (6) of Figure 3.9, concrete stem pair is shown in light green (not in M).

Notice that the cost of the precomputed stem pairs is distributed equally among the two subalignments. This is correct, because it is guaranteed that each alignment contains either both subalignments or none of them. When descending in the recursion, open stem pairs are introduced via cases (4) or (6) and are removed again via case (5).

3.4.4 Algorithms

The main part of the algorithm recursively computes costs of subalignments.

The cost of the global alignment is the value of $D(1, |S_1|; 1, |S_2||\emptyset)$. It is computed following the recursion in Eq. 3.30 with base cases $D(i, i-1; j, j-1|\emptyset) = 0$ (for all i, j). Implicitly, in each recursive step the cases involving invalid items are skipped.

Now we analyze the time and space complexities of the algorithm.

Let n be $\max(|S_1|, |S_2|)$, and let s and t be the maximal number of arcs and bases in a crossing stem, respectively.

For an item $S(a_O, a_I)$, we have $O(n^2 s^2)$ possible instances:

- for a_O: we can freely choose among the $O(n^2)$ arc pairs in CR;
- for a_I: we have $O(s^2)$ possible choices (because the arcs of a_O and a_I must belong to the same stem).

For the S' items, it is not clear when the computation of the S' items will terminate in the original paper [13]. We assume that for each S' item, the computation will terminate when the outermost arc pair of the maximal stem pair which covers a_I is encountered. Thus $S'(i, i'; j, j'; a_I)$ has $O(t^4)$ possible instances of i, i', j, j'

and $O(n^2)$ possible instances of a_I. So we need $O(n^2t^4)$ space for the S' items; the time complexity coincides with the required space, since each of these items can be computed in constant time according to the recursion in Eq. 3.27.

$D(i,i';j,j'|M)$ has $O(n^4)$ possible instances of i, i', j, j'; but only $O(n^2)$ of them need to be maintained permanently. (More precisely, we only need to maintain items $D(i+1,i'-1;j+1,j'-1|M)$ when $((i,i'),(j,j')) \in NC$.)

To measure the number of instances of M, we need the notion of the *crossing number of a position (x, y)*, defined as

$$C(x,y) = |\{(a_O, a_I) \in ST_{CR}^{MAX} \mid \nwarrow a_I \prec (x,y) \prec \searrow a_I\}|, \tag{3.31}$$

where ST_{CR}^{MAX} denotes the subset of ST_{CR} that only contains pairs of maximal stems. The maximal crossing number is denoted as k. Now we give an illustration of this notion. In the example shown in Figure 3.10, the set of crossing maximal stem pairs ST_{CR}^{MAX} is {(2,II), (2,III), (3,II)}[2]. The position (x,y) is inside the innermost arc pair of stem pair (2,II), the innermost arc pair of stem pair (2,III) and the innermost arc pair of stem pair (3,II). Thus the fixed parameter k of this example is 3.

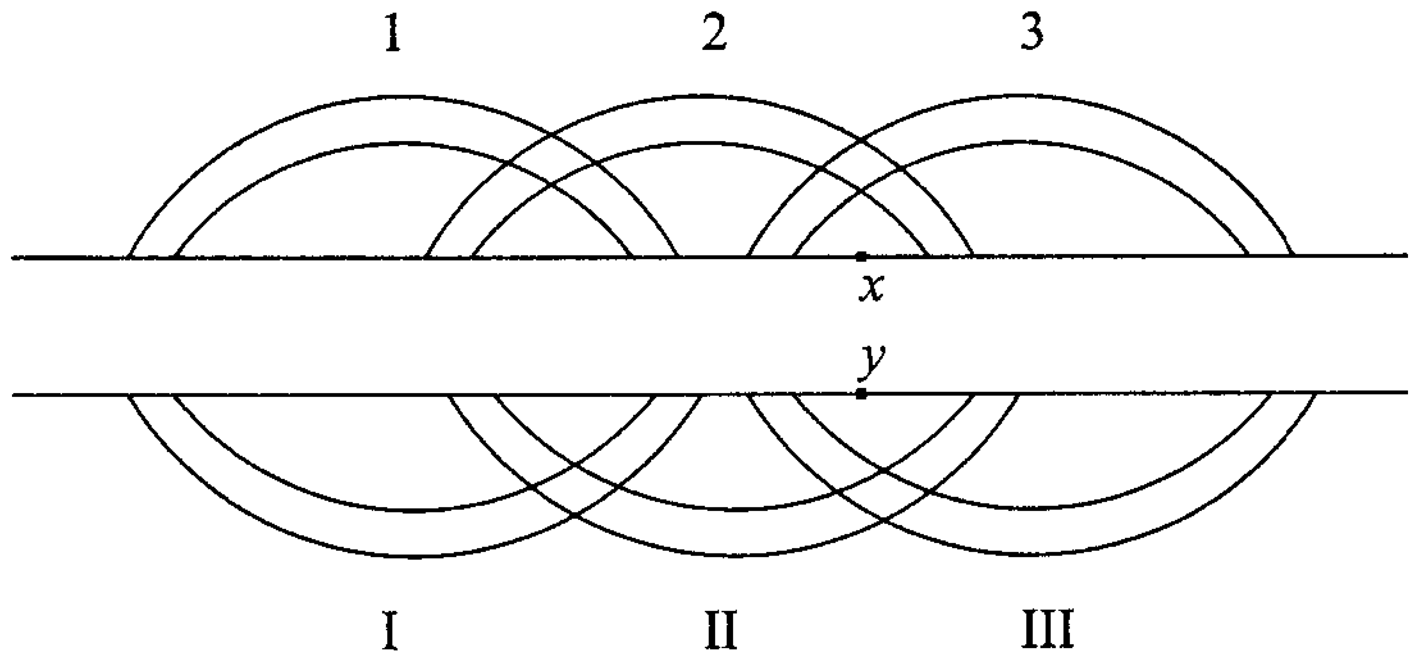


Figure 3.10: An illustration of the crossing number of a position (x, y)

Now we can measure the number of instances of M. Since each maximal stem pair has $O(s^4)$ fragments, there are at most $O((s^4)^{C(i,j)+C(i',j')}) = O(s^{8k})$ possible

[2]Similar to the partition of the set of arc pairs, we could partition the set of maximal stems to get the set of crossing maximal stem pairs and the set of non-crossing maximal stem pairs. We will discuss this in Chapter 4.

instances of M for fixed i, i', j, j'. Thus we need to compute $O(n^4 s^{8k})$ D items, and maintain $O(n^2 s^{8k})$ of them permanently.

The computation of a D item needs only for the recursive cases (4) and (6) of Eq. 3.30 more than constant time:

- for case(4): we need to iterate over all possible instances of M_1 and M_2; since M_2 is uniquely determined by M and M_1, there are $O(s^{8k})$ of these instances;
- for case (6): we need to iterate over all $O(s^2)$ possible instances of a_I.

Therefore, the computation of all the $O(n^4 s^{8k})$ D items requires $O(n^4 s^{8k} \cdot s^{8k}) = O(n^4 s^{16k})$ time.

So the space complexity of the whole algorithm is $O(n^2 s^{8k})$, and time complexity is $O(n^4 s^{16k})$.

Unfortunately, this algorithm only works if the fixed parameter k is very small, for example $k = 1$. When the parameter is large, it is not affordable due to too high usage of space and time.

Chapter 4

Improved Algorithms for Alignment between RNA Tertiary Structures

We follow the work of Möhl *et al.* [13] which has been introduced in Section 3.4. The main idea of their algorithm is partitioning the set of arc pairs $P_1 \times P_2$ into a set NC of "non-crossing" arc pairs and a set of "crossing" arc pairs $CR = P_1 \times P_2 - NC$. Then the algorithm applies a polynomial alignment method for the arc pairs in NC and an exponential alignment method for the arc pairs in CR.

The results of Möhl *et al.* [13] show that they can only compute the alignment between RNA tertiary structures if the fixed parameter k of their algorithm is very small, for example $k = 1$. Even for very simple tertiary structures, their implementation still takes too much time and space to compute optimal alignment. When the parameter is large, that is, for moderate tertiary structures, their algorithm does not work due to too high usage of space and time.

We have made several optimizations to accelerate their algorithm. For simple tertiary structures, we can compute the optimal alignment efficiently. For moderate tertiary structures, we adopt the constrained alignment approach. Although the result produced by constrained alignment is not guaranteed to be an optimal solution, in practice it would be reasonable.

4.1 Basic Definitions

In this section, we first give some definitions that will be used later. The section supplements Section 3.1.

Recall that in Section 3.1, we define a *set of tertiary arcs* of an RNA structure $R(P)$ as a subset of crossing arcs $P_{ter} \subseteq P$ which satisfies the condition that for any two arcs $p, p' \in P - P_{ter}$, p and p' do not cross. $P_{sec} = P - P_{ter}$ is a *set of secondary arcs.* If an arc $p \in P_{ter}$, then we say that p is a *tertiary arc.* If an arc $p \in P_{sec}$, then we say that p is a *secondary arc.*

We define the relation between two arc pairs as follows. Let a and a' be two arc pairs. We say that a is *before* a' if $\searrow a \prec \nwarrow a'$; alternatively, we say that a' is *after* a. We say a is *inside* a' if $\nwarrow a' \prec \nwarrow a$ and $\searrow a \prec \searrow a'$; alternatively, we say that a' is *outside* a. We say that a is *right crossed by* a' if $\nwarrow a \prec \nwarrow a' \prec \searrow a \prec \searrow a'$; alternatively, we say that a' is *left crossed by* a.

In Section 3.4.2, we introduced a new definition of "stem"[1] proposed by Möhl *et al.* [13]. In this research, we adopt their definition. We also treat a single arc as a stem. We say that a stem q *covers* an arc p if $p \in q$. A stem q is characterized by the pair (p_O, p_I) of arcs, where p_O is the outermost arc which q covers and p_I is the innermost arc which q covers.

Let $q = (p_O, p_I)$ and $q' = (p'_O, p'_I)$ be two stems. We define the relation between q and q' as follows. We say that q is *before* q' if $p_O^R < p_O'^L$; alternatively, we say that q' is *after* q. We say that q is *inside* q' if $p_I'^L < p_O^L < p_O^R < p_I'^R$; alternatively, we say that q' is *outside* q. We say that q is *crossed by* q' if $p_I^L < p_O'^L \leq p_I'^L < p_I^R \leq p_O^R < p_I'^R$ or $p_I'^L < p_O^L \leq p_I^L < p_I'^R \leq p_O'^R < p_I^R$; in the first case, q is *right crossed by* q', in the second case q is *left crossed by* q'.

[1]The definition of stem is on page 47.

An stem q is called *crossing* if it is crossed by a stem q'. If q is right crossed by q', we say that q is *right crossing*; if q is left crossed by q', we say that q is *left crossing.* A stem q is called *non-crossing* if it is not crossed by any stem q'.

A *maximal stem* (denoted as *m_stem*) q, is defined as a stem with maximal number of arcs $\{p_1, \cdots, p_k\} \subseteq P$ with $p_1^L < \cdots < p_k^L < p_k^R < \cdots < p_1^R$ such that no end of arcs in $P - q$ is in one of the intervals $[p_1^L..p_k^L]$ or $[p_k^R..p_1^R]$. We denote the set of maximal stems of an RNA structure $R(P)$ as ST^{MAX}.

We define a *set of tertiary m_stems* of an RNA structure $R(P)$ as a subset of crossing *m_stems* $ST_{ter}^{MAX} \subseteq ST^{MAX}$ which satisfies the condition that for any two *m_stems* $q, q' \in ST^{MAX} - ST_{ter}^{MAX}$, q and q' do not cross. $ST_{sec}^{MAX} = ST^{MAX} - ST_{ter}^{MAX}$ is a *set of secondary m_stems.* If a *m_stem* $q \in ST_{ter}^{MAX}$, then we say that q is a *tertiary m_stem.* If a *m_stem* $q \in ST_{sec}^{MAX}$, then we say that q is a *secondary m_stem.*

Möhl *et al.* gave a definition of stem pair in [13] which we have introduced on page 48.

Recall that we defined the partial order $\prec$ as $(x_1, y_1) \prec (x_2, y_2)$ iff $x_1 < x_2$ and $y_1 < y_2$ in Section 3.1. We now define another two partial orders. We define $\preceq$ as $(x_1, y_1) \preceq (x_2, y_2)$ iff $x_1 \leq x_2$ and $y_1 \leq y_2$. We define $\lessdot$ as $(x_1, y_1) \lessdot (x_2, y_2)$ iff $x_1 < x_2$, or $x_1 = x_2$ and $y_1 < y_2$.

We define the relation between an arc pair and a stem pair as follows. Let a be an arc pair and $c = (c_O, c_I)$ be a stem pair. We say that a is *before* c if $\searrow a \prec \nwarrow c_O$. We say that a is *after* c if $\searrow c_O \prec \nwarrow a$. We say a is *inside* c if $\nwarrow c_I \prec \nwarrow a$ and $\searrow a \prec \searrow c_I$. We say a is *outside* c if $\nwarrow a \prec \nwarrow c_O$ and $\searrow c_O \prec \searrow a$. We say that a is *right crossed by* c if $\nwarrow a \prec \nwarrow c_O$ and $\nwarrow c_I \prec \searrow a \prec \searrow c_I$. We say that a is *left crossed by* c if $\nwarrow c_I \prec \nwarrow a \prec \searrow c_I$ and $\searrow c_O \prec \searrow a$.

We define the relation between two stem pairs as follows. Let $b = (b_O, b_I)$ and $c = (c_O, c_I)$ be two stem pairs. We say that b is *before* c if $\searrow b_O \prec \nwarrow c_O$; alternatively, we say that c is *after* b. We say that b is *inside* c if $\nwarrow c_I \prec \nwarrow b_O$ and $\searrow b_O \prec \searrow c_I$; alternatively, we say that c is *outside* b. We say that b is *right crossed by* c if $\nwarrow b_I \prec \nwarrow c_O$, $\nwarrow c_I \prec \searrow b_I$ and $\searrow b_O \prec \searrow c_I$; alternatively, we say that c is *left crossed by*

b.

Möhl *et al.* gave a definition that a stem pair covers an arc pair in [13] which we have introduced on page 48. We say that a stem pair (a_{O_1}, a_{I_1}) is *covered* by another stem pair (a_{O_2}, a_{I_2}) if its outermost arc pair a_{O_1} and innermost arc pair a_{I_1} are covered by (a_{O_2}, a_{I_2}).

Similar to the partition method of arc pairs $P_1 \times P_2$ into NC and CR, we can partition the set of *m_stem* pairs $ST_1^{MAX} \times ST_2^{MAX}$ into a set of crossing *m_stem* pairs and a set of non-crossing *m_stem* pairs. We denote the set of crossing *m_stem* pairs as ST_{CR}^{MAX}, and the set of non-crossing *m_stem* pairs as ST_{NC}^{MAX}.

We use ST_{CR}^{all} to denote the set of all crossing stem pairs (a_O, a_I) covered by crossing *m_stem* pairs in ST_{CR}^{MAX}.

A stem pair $(a_O, a_I) \in ST_{CR}^{all}$ is *open for a subalignment* $(R'_1, R'_2)[i_1, i_2; j_1, i_2]$ *in an alignment* (R'_1, R'_2) iff a_O and a_I are open for $(R'_1, R'_2)[i_1, i_2; j_1, i_2]$.

The *set of proper open stem pairs of a subalignment* $(R'_1, R'_2)[i_1, i_2; j_1, i_2]$ *in an alignment* (R'_1, R'_2) is a set M of open stem pairs of $(R'_1, R'_2)[i_1, i_2; j_1, i_2]$ such that no two stem pairs in M are covered by the same *m_stem* pair. The reason for this definition is that for two open stem pairs $b = (b_O, b_I)$ and $c = (c_O, c_I)$ which are covered by the same *m_stem* pair s, we can actually substitute b and c with another stem pair d that is covered by s ($d = (b_O, c_I)$ if b is outside c; $d = (c_O, b_I)$ if c is outside b).

We call the stem pair (a_O, a_I) *local maximal* if it is maximal among all stem pairs of which innermost arc pair is a_I or it is maximal among all stem pairs of which outermost arc pair is a_O; in the first case, (a_O, a_I) is called *inner local maximal*, in the second case (a_O, a_I) is called *outer local maximal.*

We say that an arc pair a is *compatible with* an arc pair a' if a is inside a', or a is outside a', or a is before a', or a is after a', or a is left crossed by a', or a is right crossed by a'.

We say that an arc pair a is *compatible with* an arc pairs set A' if and only if $\forall a' \in A'$, a is compatible with a'. Notice that if $A' = \emptyset$, we also say that a is compatible with A'.

We say that an arc pairs set A is *compatible with* an arc pairs set A' if and only if $\forall a \in A, a' \in A'$, a is compatible with a'. Notice that if $A = \emptyset$ or $A' = \emptyset$, we also say that A is compatible with A'.

We say that an arc pair a is *compatible with* a stem pair c if a is inside c, or a is outside c, or a is before c, or a is after c, or a is left crossed by c, or a is right crossed by c.

We say that an arc pair a is *compatible with* a stem pairs set C if and only if $\forall c \in C$, a is compatible with c. Notice that if $C = \emptyset$, we also say that a is compatible with C.

We say that a crossing stem pair $b \in ST_{CR}^{all}$ is *compatible with* a crossing stem pair $c \in ST_{CR}^{all}$ if b is inside c, and b, c are not covered by the same crossing *m_stem* pair; or b is outside c, and b, c are not covered by the same crossing *m_stem* pair; or b is before c; or b is after c; or b is left crossed by c; or b is right crossed by c. The condition that b, c are not covered by the same crossing *m_stem* pair in first two cases are required by the property of proper open stem pairs set.

We say that a crossing stem pair $b \in ST_{CR}^{all}$ is *compatible with* a crossing stem pairs set $C \subseteq ST_{CR}^{all}$ if and only if $\forall c \in C$, b is compatible with c. Notice that if $C = \emptyset$, we also say that b is compatible with C.

We say that a crossing stem pairs set $B \subseteq ST_{CR}^{all}$ is *compatible with* a crossing stem pairs set $C \subseteq ST_{CR}^{all}$ if and only if $\forall b \in B, c \in C$, b is compatible with c. Notice that if $B = \emptyset$ or $C = \emptyset$, we also say that B is compatible with C.

4.2 Partition Arc Pairs into Crossing Arc Pairs and Non-crossing Arc Pairs

In Section 3.4.1, we introduced two partition methods of arc pairs set given in [13]. In this section, We propose a new partition approach (two partition methods) according to tertiary arcs and crossing arcs, and adopt the greedy strategy to make further optimization.

The first partition method is according to tertiary arcs of the first RNA and crossing arcs of the second RNA.

Lemma 4.2.1 *The partition of $P_1 \times P_2$ into $CR = CR_1 \times CR_2 = \{p_1 \in P_1 \mid p_1$ is tertiary$\} \times \{p_2 \in P_2 \mid p_2$ is crossing$\}$ and $NC = P_1 \times P_2 - CR$ is valid.*

Proof: We have

$$\begin{aligned}
NC &= P_1 \times P_2 - CR \\
&= (CR_1 \cup (P_1 - CR_1)) \times (CR_2 \cup (P_2 - CR_2)) - CR_1 \times CR_2 \\
&= ((P_1 - CR_1) \times CR_2) \cup (CR_1 \times (P_2 - CR_2)) \cup ((P_1 - CR_1) \times (P_2 - CR_2)) \\
&= ((P_1 - CR_1) \times CR_2) \cup (P_1 \times (P_2 - CR_2))
\end{aligned}$$

where $P_1 - CR_1$ is the set of secondary arcs of the first RNA and $P_2 - CR_2$ is the set of non-crossing arcs of the second RNA.

From the definition of crossing arc pairs in Section 3.1, we know that two arc pair $a = (a_1, a_2)$ and $b = (b_1, b_2)$ cross if and only if the arc a_1 is left crossed by the arc b_1 and the arc a_2 is left crossed by the arc b_2, or the arc a_1 is right crossed by the arc b_1 and the arc a_2 is right crossed by the arc b_2.

No arc pair in $P_1 \times P_2$ cross arc pairs in $P_1 \times (P_2 - CR_2)$, since arcs in $P_2 - CR_2$ are non-crossing.

For an arc pair $c = (p_1, p_2) \in (P_1 - CR_1) \times CR_2$, we have the following cases.

- p_1 is non-crossing. Thus no arc pair in $P_1 \times P_2$ crosses c.
- p_1 is crossing. For this case, we have three subcases. (1) No arc left crosses p_1 (i.e. all arcs crossing p_1 right cross p_1) and no arc right crosses p_2 (i.e. all arcs crossing p_2 left cross p_2), thus no arc pair in $P_1 \times P_2$ crosses c. (2) No arc right crosses p_1 and no arc left crosses p_2. Similar to subcase (1), no arc pair in

$P_1 \times P_2$ crosses c. (3) All other cases. There must be some arc pair crossing c. Since all arcs that cross p_1 must be in CR_1 and all arcs that cross p_2 must be in CR_2, all arc pairs that cross $c = (p_1, p_2)$ must be in $CR_1 \times CR_2$, that is CR.

Therefore, for all $c, c' \in NC$ it holds that c and c' do not cross. Hence the lemma holds. □

Thus a valid partition can be obtained if CR_1 contains all tertiary arcs of the first RNA and CR_2 contains all crossing arcs of the second RNA. Analogously, a valid partition can be obtained if CR_1 contains all crossing arcs of the first RNA and CR_2 contains all tertiary arcs of the second RNA.

Lemma 4.2.2 *The partition of $P_1 \times P_2$ into $CR = CR_1 \times CR_2 = \{p_1 \in P_1 \mid p_1$ is crossing$\} \times \{p_2 \in P_2 \mid p_2$ is tertiary$\}$ and $NC = P_1 \times P_2 - CR$ is valid.*

Proof: Similar to Lemma 4.2.1. □

In the example shown in Figure 3.5 in Section 3.1, P_1 = {1, 2, 3} and P_2 = {I, II, III}. Applying Lemma 4.2.1 to the example, we can get CR = {2} × {I, II, III} = {(2, I), (2, II), (2, III)} and NC = {(1, I), (1, II), (1, III), (3, I), (3, II), (3, III)}. Applying Lemma 4.2.2 to the example, we can get CR = {1, 2, 3} × {II} = {(1, II), (2, II), (3, II)} and NC = {(1, I), (2, I), (3, I), (1, III), (2, III), (3, III)}.

The partition according to Lemma 4.2.1 or Lemma 4.2.2 sometimes is not local minimal.

We can use the following method to make further optimization.

For each partition method we have discussed (Lemma 3.4.2, Lemma 3.4.3, Lemma 4.2.1 and Lemma 4.2.2), we use the result of the method as a starting point. We check each element of CR against all elements of NC; if it does not cross any arc pair in current NC,then we move it to NC. We continue to do this step until all elements of CR have been checked. Then we get a local minimal partition.

For the example shown in Figure 3.5 in Section 3.1, the partition according to Lemma 3.4.2 is CR = {(2, II), (2, III), (3, II), (3, III)} and NC = {(1, I), (1, II), (1, III), (2, I), (3, I)}. Applying the optimization method to the result, we can move (3, III) from CR to NC and get the final result $CR = \{(2, \text{II}), (2, \text{III}), (3, \text{II})\}$ and $NC = \{(1, \text{I}), (1, \text{II}), (1, \text{III}), (2, \text{I}), (3, \text{I}), (3, \text{III})\}$.

We select the best partition, i.e. the partition which has the smallest $|CR|$, from optimized results produced by the four partition methods for later use.

4.3 A General Score Scheme

We now describe a general score scheme for computing alignment scores.

In this research, we will not consider an explicit base-pair altering operation introduced in Section 3.1 since that operation is replaced by a base-pair bond breaking operation and then an unpaired base deletion operation. We adopt the linear gap penalty model.

Consider two RNA structures R_1 and R_2, we use $\Gamma(\ )$ introduced in Section 3.1 to define $\gamma(i,j)$ for $0 \le i \le |R_1|$ and $0 \le j \le |R_2|$, and $\delta((i',i),(j',j))$ where $(R_1[i'], R_1[i])$ and $(R_2[j'], R_2[j])$ are base pairs in R_1 and R_2, respectively. We distribute evenly the deletion/insertion/bond breaking cost of a base pair to its two bases.

If $i = p_{r_1}(i)$ and $j = p_{r_2}(j)$,

$$\gamma(i,0) = \Gamma(R_1[i] \to \lambda) \tag{4.1}$$

$$\gamma(0,j) = \Gamma(\lambda \to R_2[j]) \tag{4.2}$$

$$\gamma(i,j) = \Gamma(R_1[i] \to R_2[j]) \tag{4.3}$$

If $i' = p_{r_1}(i) < i$,

$$\gamma(i',0) = \Gamma((R_1[i'], R_1[i]) \to \lambda)/2 \tag{4.4}$$

$$\gamma(i,0) = \Gamma((R_1[i'], R_1[i]) \to \lambda)/2 \tag{4.5}$$

If $j' = p_{r_2}(j) < j$,

$$\gamma(0, j') = \Gamma(\lambda \to (R_2[j'], R_2[j]))/2 \tag{4.6}$$

$$\gamma(0, j) \ = \Gamma(\lambda \to (R_2[j'], R_2[j]))/2 \tag{4.7}$$

If $i = p_{r_1}(i)$ and $j' = p_{r_2}(j) < j$,

$$\gamma(i, j') = \Gamma_b((R_2[j'], R_2[j]))/2 + \Gamma(R_1[i] \to R_2[j']) \tag{4.8}$$

$$\gamma(i, j) \ = \Gamma_b((R_2[j'], R_2[j]))/2 + \Gamma(R_1[i] \to R_2[j]) \tag{4.9}$$

If $i' = p_{r_1}(i) < i$ and $j = p_{r_2}(j)$,

$$\gamma(i', j) = \Gamma_b((R_1[i'], R_1[i]))/2 + \Gamma(R_1[i'] \to R_2[j]) \tag{4.10}$$

$$\gamma(i, j) \ = \Gamma_b((R_1[i'], R_1[i]))/2 + \Gamma(R_1[i] \to R_2[j]) \tag{4.11}$$

If $i' = p_{r_1}(i) < i$ and $j' = p_{r_2}(j) < j$,

$$\begin{aligned} \gamma(i', j') &= \Gamma_b((R_1[i'], R_1[i]))/2 + \Gamma_b((R_2[j'], R_2[j]))/2 \\ &\quad + \Gamma(R_1[i'] \to R_2[j']) \end{aligned} \tag{4.12}$$

$$\begin{aligned} \gamma(i', j) &= \Gamma_b((R_1[i'], R_1[i]))/2 + \Gamma_b((R_2[j'], R_2[j]))/2 \\ &\quad + \Gamma(R_1[i'] \to R_2[j]) \end{aligned} \tag{4.13}$$

$$\begin{aligned} \gamma(i, j') &= \Gamma_b((R_1[i'], R_1[i]))/2 + \Gamma_b((R_2[j'], R_2[j]))/2 \\ &\quad + \Gamma(R_1[i] \to R_2[j']) \end{aligned} \tag{4.14}$$

$$\begin{aligned} \gamma(i, j) &= \Gamma_b((R_1[i'], R_1[i]))/2 + \Gamma_b((R_2[j'], R_2[j]))/2 \\ &\quad + \Gamma(R_1[i] \to R_2[j]) \end{aligned} \tag{4.15}$$

$$\delta((i', i), (j', j)) = \Gamma((R_1[i'], R_1[i]) \to (R_2[j'], R_2[j])) \tag{4.16}$$

From this definition, if $R_1[i]$ is a single base, then $\gamma(i, 0)$ is the cost of deleting this base and if $R_1[i]$ is a base of a base pair, then $\gamma(i, 0)$ is half of the cost of deleting this base pair. The meaning of $\gamma(0, j)$ is similar. If both $R_1[i]$ and $R_2[j]$ are single bases, then $\gamma(i, j)$ is the cost of aligning bases $R_1[i]$ and $R_2[j]$. If $R_1[i]$ is a single base and $R_2[j]$ is a base of a base pair, then $\gamma(i, j)$ is half of the bond breaking cost of the base pair involving $R_2[j]$ plus the cost of aligning bases $R_1[i]$ and $R_2[j]$. If $R_1[i]$ is a base of a base pair and $R_2[j]$ is a single base, then $\gamma(i, j)$ is half of the bond breaking cost of the base pair involving $R_1[i]$ plus the cost of aligning bases $R_1[i]$ and $R_2[j]$. If both $R_1[i]$ and $R_2[j]$ are bases of base pairs, then $\gamma(i, j)$ is half of the bond breaking cost of the base pair involving $R_1[i]$ plus half of the bond breaking cost of the base pair involving $R_2[j]$ plus the cost of aligning bases $R_1[i]$ and $R_2[j]$. This is used to deal with the cases where the two base pairs involving $R_1[i]$ and $R_2[j]$ are not aligned. $\delta((i', i), (j', j))$ is the cost of aligning base pairs $(R_1[i'], R_1[i])$ and $(R_2[j'], R_2[j])$.

Obviously, our score scheme is an extension of the score scheme of Wang and Zhang's algorithm introduced on page 38. It is more general than the score scheme of Möhl *et al.*'s algorithm introduced on page 45.

4.4 Property of Optimal Alignments

In this section, we consider the property of optimal alignment between two RNA tertiary structures.

We consider the optimal alignment between $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$. We use $A(i_1, i; j_1, j|N)$ to represent the optimal alignment cost between $R_1[i_1, i]$ and $R_2[j_1, j]$ where $N \subseteq CR$ is the set of open arc pairs[2] of the optimal alignment.

We can now consider how to compute the optimal alignment between $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$.

Lemma 4.4.1

$$A(\emptyset; \emptyset|\emptyset) = 0 \tag{4.17}$$

[2]The definition of open arc pairs set is on page 30.

Proof: Consider $A(i_1, i_1; j_1, j_1|\emptyset)$ where $R_1[i_1]$ and $R_2[j_1]$ are single bases. If the optimal alignment results from aligning $R_1[i_1]$ to $R_2[j_1]$, then we only need to account for the cost for aligning $R_1[i_1]$ to $R_2[j_1]$. Hence we may set $A(\emptyset; \emptyset|\emptyset) = 0$. $\square$

Lemma 4.4.2 *For $i_1 \leq i \leq i_2$,*

$$A(i_1, i; \emptyset|\emptyset) = A(i_1, i-1; \emptyset|\emptyset) + \gamma(i, 0) \tag{4.18}$$

For $j_1 \leq j \leq j_2$,

$$A(\emptyset; j_1, j|\emptyset) = A(\emptyset; j_1, j-1|\emptyset) + \gamma(0, j) \tag{4.19}$$

Proof: For $A(i_1, i; \emptyset|\emptyset)$, it is obvious that each element in $R_1[i_1, i]$ is aligned to $'-'$. That is, $R_1[i]$ is aligned to $'-'$, each element in $R_1[i_1, i-1]$ is aligned to $'-'$ and the open arc pair set is still $\emptyset$. Hence we have $A(i_1, i; \emptyset|\emptyset) = A(i_1, i-1; \emptyset|\emptyset) + \gamma(i, 0)$.

Similarly, we can obtain the other formula. $\square$

Lemma 4.4.3 *For $i_1 \leq i \leq i_2$ and $j_1 \leq j \leq j_2$,*

$$
A(i_1,i;j_1,j|N) =
$$

$$
\min \begin{cases}
\textit{skip (1) if there exists some arc pair } a \in N \textit{ with } \nwarrow a = (i,j') \textit{ or } \searrow a = (i,j') \\
\quad \textit{where } j_1 \leq j' \leq j \\
\quad A(i_1,i-1;j_1,j|N) + \gamma(i,0) & \textit{(1)} \\
\textit{skip (2) if there exists some arc pair } a \in N \textit{ with } \nwarrow a = (i',j) \textit{ or } \searrow a = (i',j) \\
\quad \textit{where } i_1 \leq i' \leq i \\
\quad A(i_1,i;j_1,j-1|N) + \gamma(0,j) & \textit{(2)} \\
\textit{skip (3) if there exists some arc pair } a \in N \textit{ with } \nwarrow a = (i,j') \textit{ or } \searrow a = (i,j') \\
\quad \textit{where } j_1 \leq j' \leq j, \textit{ or } \nwarrow a = (i',j) \textit{ or } \searrow a = (i',j) \textit{ where } i_1 \leq i' \leq i \\
\quad A(i_1,i-1;j_1,j-1|N) + \gamma(i,j) & \textit{(3)} \\
\textit{if } (i_1,j_1) \preceq (p_{r_1}(i),p_{r_2}(j)) \prec (i,j) \textit{ and } ((p_{r_1}(i),i),(p_{r_2}(j),j)) \in NC, \\
\quad \textit{and } ((p_{r_1}(i),i),(p_{r_2}(j),j)) \textit{ is compatible with } N \\
\quad \min\limits_{(N_1,N_2)} \left\{ \begin{array}{c|l} & N_1,N_2 \subseteq CR, \textit{ where} \\ A(i_1,p_{r_1}(i)-1;j_1,p_{r_2}(j)-1|N_1) & N_{share} = N_1 \cap N_2, \\ +\,A(p_{r_1}(i)+1,i-1;p_{r_2}(j)+1,j-1|N_2) & N = (N_1 \cup N_2) - N_{share} \\ +\,\delta((p_{r_1}(i),i),(p_{r_2}(j),j)) & \textit{and } N_{share} \textit{ is} \\ & \textit{compatible with } N \end{array} \right\} & \textit{(4)} \\
\textit{if there exists some arc pair } a \in N \textit{ with } \nwarrow a = (i,j) \\
\quad A(i_1,i-1;j_1,j-1|N-\{a\}) + \delta(i,(p_{r_1}(i)),(j,p_{r_2}(j)))/2 & \textit{(5.a)} \\
\textit{if there exists some arc pair } a \in N \textit{ with } \searrow a = (i,j) \\
\quad A(i_1,i-1;j_1,j-1|N-\{a\}) + \delta((p_{r_1}(i),i),(p_{r_2}(j),j))/2 & \textit{(5.b)} \\
\textit{if there exists some arc pair } a \in CR \textit{ with} \\
\quad \searrow a = (i,j) \textit{ and } (i_1,j_1) \preceq \nwarrow a = (p_{r_1}(i),p_{r_2}(j)), \textit{ and } a \textit{ is compatible with } N \\
\quad A(i_1,i-1;j_1,j-1|N \cup \{a\}) + \delta((p_{r_1}(i),i),(p_{r_2}(j),j))/2 & \textit{(6)}
\end{cases}
\tag{4.20}
$$

Proof: Consider $R_1[i]$ and $R_2[j]$. There are exactly the following cases.

(1) $R_1[i]$ is aligned to '$-$'. Thus $R_1[i_1, i-1]$ is aligned to $R_2[j_1, j]$ and the open arc pairs set is still N. Hence the $A(i_1, i-1; j_1, j|N) + \gamma(i, 0)$ item. Notice that if there exists some arc pair $a \in N$ with $\nwarrow a = (i, j')$ or $\searrow a = (i, j')$ where $j_1 \le j' \le j$, then $R_1[i]$ must be aligned to $R_2[j']$ since a is an open arc pair and realized in the alignment. Thus for these situations, we need to skip case (1).

(2) $R_2[j']$ is aligned to '$-$'. Similar to case (1). Notice that if there exists some arc pair $a \in N$ with $\nwarrow a = (i', j)$ or $\searrow a = (i', j)$ where $i_1 \le i' \le i$, then $R_2[j]$ must be aligned to $R_1[i']$ since a is an open arc pair and realized in the alignment. Thus for these situations, we need to skip case (2).

(3) $R_1[i]$ is aligned to $R_2[j]$, but no arc pair involving $(R_1[i], R_2[j])$ is realized. Thus $R_1[i_1, i-1]$ is aligned to $R_2[j_1, j-1]$ and the open arc pairs set is still N. Hence the $A(i_1, i-1; j_1, j-1|N) + \gamma(i, j)$ item. Notice that if there exists some arc pair $a \in N$ with $\nwarrow a = (i, j')$ or $\searrow a = (i, j')$ where $j_1 \le j' \le j$, then $R_1[i]$ must be aligned to $R_2[j']$ and a must be realized in the alignment since a is an open arc pair; if there exists some arc pair $a \in N$ with $\nwarrow a = (i', j)$ or $\searrow a = (i', j)$ where $i_1 \le i' \le i$, then $R_2[j]$ must be aligned to $R_1[i']$ and a must be realized in the alignment since a is an open arc pair. Thus for these situations, we need to skip case (3).

(4) $R_1[i]$ is aligned to $R_2[j]$, and the arc pair $((p_{r_1}(i), i), (p_{r_2}(j), j))$ is realized. This requires that $((p_{r_1}(i), i), (p_{r_2}(j), j))$ is compatible with N. Then the optimal alignment between $R_1[i_1, i]$ and $R_2[j_1, j]$ is partitioned into three parts: 1. the optimal alignment between $R_1[i_1, p_{r_1}(i) - 1]$ and $R_2[j_1, p_{r_2}(j) - 1]$, 2. the optimal alignment between $R_1[p_{r_1}(i) + 1, i-1]$ and $R_2[p_{r_2}(j) + 1, j-1]$, and 3. the alignment of $(R_1[p_{r_1}(i)], R_1[i])$ to $(R_2[p_{r_2}(j)], R_2[j])$. For part 1 and part 2, we denote their corresponding open arc pairs sets by N_1 and N_2, respectively. Hence the $A(i_1, p_{r_1}(i) - 1; j_1, p_{r_2}(j) - 1|N_1) + A(p_{r_1}(i) + 1, i-1; p_{r_2}(j) + 1, j-1|N_2) + \delta((p_{r_1}(i), i), (p_{r_2}(j), j))$ item. N_1 and N_2 may contain open arc pairs which are not contained in N. We use N_{share} to denote the set of arc pairs that are shared between part 1 and part 2, i.e. $N_{share} = N_1 \cap N_2$. N, N_1, N_2, N_{share} satisfy that $N = (N_1 \cup N_2) - N_{share}$ and N_{share} is compatible with

N. We need to minimize over all possible alternatives.

(5) We have two subcases. (a) An arc pair $a \in N$ has left end in (i, j). Then the optimal alignment between $R_1[i_1, i]$ and $R_2[j_1, j]$ is partitioned into two parts: 1. the optimal alignment between $R_1[i_1, i-1]$ and $R_2[j_1, j-1]$, 2. the alignment of $R_1[i]$ and $R_2[j]$. For part 1, a is no more open, thus its open arc pairs set is $N - \{a\}$, and the alignment cost is $A(i_1, i-1; j_1, j-1|N - \{a\})$. For part 2, the cost is half of the cost of aligning base pair $(R_1[i], R_1[(p_{r_1}(i)])$ to base pair $(R_2[j], R_2[(p_{r_2}(j)])$. (b) An arc pair $a \in N$ has right end in (i, j). It is similar to subcase (a).

(6) An arc pair a in CR has right end in (i, j), a is compatible with N, and a is not open for the alignment between $R_1[i_1, i]$ and $R_2[j_1, j]$. Then the optimal alignment between $R_1[i_1, i]$ and $R_2[j_1, j]$ is partitioned into two parts: 1. the optimal alignment between $R_1[i_1, i-1]$ and $R_2[j_1, j-1]$, 2. the alignment of $R_1[i]$ and $R_2[j]$. For part 1, a is now open, thus its open arc pairs set is $N \cup \{a\}$, and the alignment cost is $A(i_1, i-1; j_1, j-1|N \cup \{a\})$. For part 2, the cost is half of the cost of aligning base pair $(R_1[(p_{r_1}(i)], R_1[i])$ to base pair $(R_2[(p_{r_2}(j)], R_2[j])$.

Therefore we take the minimum of all the cases and get the above recursion. $\square$

4.5 Algorithms

From Lemma 4.4.1 to 4.4.3, we can compute optimal alignment cost $A(1, |R_1|; 1, |R_2||\emptyset)$ between R_1 and R_2 using a bottom-up approach. However, the complexity of the algorithm will be too high if we directly use these lemmas, since we need to enumerate all the combinations of crossing arc pairs. We need to consider some methods to accelerate computation. We will show that we can preprocess crossing stem pairs in ST_{CR}^{all} to accelerate computation in Section 4.5.2.

Before we discuss the preprocessing of crossing stem pairs in ST_{CR}^{all}, we need to show how to compute the set ST_{CR}^{MAX} of crossing *m_stem* pairs and the set ST_{NC}^{MAX} of non-crossing *m_stem* pairs that will be used later.

4.5.1 Partition *m_stem* Pairs into Crossing *m_stem* Pairs and Non-crossing *m_stem* Pairs

Similar to the partition method of arc pairs $P_1 \times P_2$ into NC and CR, we can partition the set of *m_stem* pairs $ST_1^{MAX} \times ST_2^{MAX}$ into a set ST_{CR}^{MAX} of crossing *m_stem* pairs and a set ST_{NC}^{MAX} of non-crossing *m_stem* pairs.

We extend Definition 3.4.1 to *m_stem* pairs, and the four partition methods of arc pairs to *m_stem* pairs. Lemma 4.5.2 to 4.5.5 can be easily proved by similar technique used in the proofs of lemmas for partition of arc pairs.

Definition 4.5.1 *A partition of $ST_1^{MAX} \times ST_2^{MAX}$ into ST_{NC}^{MAX} and ST_{CR}^{MAX} is valid if and only if for all $b, b' \in ST_{NC}^{MAX}$ it holds that b and b' do not cross.*

Lemma 4.5.2 *The partition of $ST_1^{MAX} \times ST_2^{MAX}$ into $ST_{CR}^{MAX} = \{a_1 \in ST_1^{MAX} \mid a_1$ is left crossing$\} \times \{a_2 \in ST_2^{MAX} \mid a_2$ is left crossing$\}$ and $ST_{NC}^{MAX} = ST_1^{MAX} \times ST_2^{MAX} - ST_{CR}^{MAX}$ is valid.*

Lemma 4.5.3 *The partition of $ST_1^{MAX} \times ST_2^{MAX}$ into $ST_{CR}^{MAX} = \{a_1 \in ST_1^{MAX} \mid a_1$ is right crossing$\} \times \{a_2 \in ST_2^{MAX} \mid a_2$ is right crossing$\}$ and $ST_{NC}^{MAX} = ST_1^{MAX} \times ST_2^{MAX} - ST_{CR}^{MAX}$ is valid.*

Lemma 4.5.4 *The partition of $ST_1^{MAX} \times ST_2^{MAX}$ into $ST_{CR}^{MAX} = \{a_1 \in ST_1^{MAX} \mid a_1$ is tertiary$\} \times \{a_2 \in ST_2^{MAX} \mid a_2$ is crossing$\}$ and $ST_{NC}^{MAX} = ST_1^{MAX} \times ST_2^{MAX} - ST_{CR}^{MAX}$ is valid.*

Lemma 4.5.5 *The partition of $ST_1^{MAX} \times ST_2^{MAX}$ into $ST_{CR}^{MAX} = \{a_1 \in ST_1^{MAX} \mid p_1$ is crossing$\} \times \{a_2 \in ST_2^{MAX} \mid a_2$ is tertiary$\}$ and $ST_{NC}^{MAX} = ST_1^{MAX} \times ST_2^{MAX} - ST_{CR}^{MAX}$ is valid.*

We can also make further optimization as we do in Section 4.2. For each partition method (Lemma 4.5.2, Lemma 4.5.3, Lemma 4.5.4 and Lemma 4.5.5), we use the result of the method as a starting point. We check each element of ST_{CR}^{MAX} against all elements of ST_{NC}^{MAX}; if it does not cross any *m_stem* pair in current ST_{NC}^{MAX}, then we move it to ST_{NC}^{MAX}. We continue to do this step until all elements of ST_{CR}^{MAX} have been checked. Then we get a local minimal partition.

Assume that we already have sorted (by $5'$ end) list of *m_stems* and sorted (by $5'$ end) lists of tertiary *m_stems* of two RNA structures R_1 and R_2[3]. We first compute a sorted (by $5'$ ends) list of *m_stem* pairs, sorted (by $5'$ end) lists of left crossing *m_stems* of two RNAs, sorted (by $5'$ end) lists of right crossing *m_stems* of two RNAs and sorted (by $5'$ end) lists of crossing *m_stems* of two RNAs.

Then we can partition the list of *m_stem* pairs into a list of crossing *m_stem* pairs and a list of non-crossing *m_stem* pairs according to Lemma 4.5.2 to 4.5.5, respectively. For the four partition results, we use the method we discussed previously to make further optimization. Thus we will get four local minimal partitions. We need to select the best partition from these four results. As we will see in Section 4.5.3, the number of crossing stem pairs, that is $|ST_{CR}^{all}|$, highly influences the complexity. Thus we want to choose the partition which will produce the least crossing stem pairs.

Actually, we can precompute the cardinality of ST_{CR}^{all} without generating ST_{CR}^{all}. For each stem b, we can save the number of arcs that it covers along with it and denote this number as $b.size$. Then we can easily evaluate the cardinality of ST_{CR}^{all} via ST_{CR}^{MAX}. We can do this as follows. For each crossing *m_stem* pair $(b_1, b_2) \in ST_{CR}^{MAX}$ where b_1 is a *m_stem* from the first RNA and b_2 is a *m_stem* from the second RNA[4], we can generate two types of crossing stem pairs: (1) we can select one arc from b_1 and select one arc b_2 such that these two arcs form a crossing arc pair which can be considered as a crossing stem pair; (2) we can select two arcs from b_1 and select two arcs b_2 such that these arcs form a crossing stem pair. The number of arc pairs of the first type is $b_1.size \cdot b_2.size$, and the number of arc pairs of the second type

[3]As we will see in Chapter 5, the input file contains primary, secondary and tertiary structures information of two input RNAs. From that information, we can easily obtain these lists.

[4]This is different from the original representation of stem pairs where we use the outermost arc pair and the innermost arc pair. We only use this notation in this section to explain how to precompute the cardinality of ST_{CR}^{all} and generate NC.

is $\binom{b_1.size}{2} \cdot \binom{b_2.size}{2}$. Thus for each crossing *m_stem* pair (b_1, b_2), we can generate $b_1.size \cdot b_2.size + \binom{b_1.size}{2} \cdot \binom{b_2.size}{2}$ crossing stem pairs. Therefore from ST_{CR}^{MAX}, we can generate $\sum_{(b_1,b_2)\in ST_{CR}^{MAX}} (b_1.size \cdot b_2.size + \binom{b_1.size}{2} \cdot \binom{b_2.size}{2})$ crossing stem pairs in total. For each partition result, we compute the number of crossing stem pairs that it can generate. Then we select the partition which will produce the least crossing stem pairs.

In the final partition, we have a sorted list ST_{CR}^{MAX}[5] of crossing *m_stem* pairs and a sorted list ST_{NC}^{MAX} of non-crossing *m_stem* pairs. These two lists are sorted by $5'$ ends of two RNAs. More precisely, for stem pairs $(a_O, a_I) \in ST_{CR}^{MAX}$ or $(a_O, a_I) \in ST_{NC}^{MAX}$, (a_O, a_I) is already sorted by the left end of the outermost arc pair $\nwarrow a_O = (i, j)$ according to the partial order $\lessdot$ defined in Section 4.1.

However, as we will see in Section 4.5.3, we need a sorted list of non-crossing *m_stem* pairs which is sorted by $3'$ ends. Thus we need to sort the list ST_{NC}^{MAX} by $3'$ ends. For stem pairs $(a_O, a_I) \in ST_{NC}^{MAX}$, we sort (a_O, a_I) by the right end of the outermost arc pair $\searrow a_O = (i, j)$ according to the partial order $\lessdot$.

As as we will see in Section 4.5.3, we also need a sorted list NC of non-crossing arc pairs which is sorted by $3'$ ends. It can be easily generated from ST_{NC}^{MAX}. For each non-crossing *m_stem* pair $(b_1, b_2) \in ST_{NC}^{MAX}$ where b_1 is a *m_stem* in the first RNA and b_2 is a *m_stem* in the second RNA, we can select one arc from b_1 and select one arc b_2, then these two arcs will form a non-crossing arc pair. We enumerate all possible alternatives, then get NC. Then we sort NC by $3'$ ends of two RNAs. For arc pairs $a \in NC$, we sort a by the right end $\searrow a = (i, j)$ according to the partial order $\lessdot$.

[5]When there is no confusion, we use the same notation to denote a set and its corresponding sorted list in the remaining part of this thesis. This is just for simplicity.

4.5.2 Accelerating Computation by Preprocessing Crossing Stem Pairs

In this section, we consider how to accelerate computation by preprocessing of crossing stem pairs in ST_{CR}^{all}. This section is based on Section 3.4.2. We propose some new methods. We present a method to filter out unnecessary "crossing stem pairs" to accelerate the computation of optimal alignment. We gave a definition of *local m_stem pair* (including *inner local m_stem pair* and *outer local m_stem pair*) in Section 4.1. In this section, we will show that we only need to compute the alignment of crossing local *m_stem* pairs. Alignment costs of other crossing stem pairs would be byproducts.

Recall that a stem pair is denoted by (a_O, a_I), where a_O is its outermost arc pair and a_I is its innermost arc pair. The cost of aligning a crossing stem pair $(a_O, a_I) \in ST_{CR}^{all}$ is the cost to align bases in the stem, and is denoted as $S(a_O, a_I)$. Recall that a stem pair (a_O, a_I) is realized in an alignment (R_1', R_2') if and only if both a_O and a_I are realized in (R_1', R_2'). Realized crossing stem pairs serve as open stem pairs for some subalignments in the core algorithm. So we want to consider only crossing stem pairs which are realized.

Möhl *et al.* proposed a approach to compute $S(a_O, a_I)$ in [13], which we have described in Section 3.4.2. They let a_I be realized and computed $S(a_O, a_I)$ following the recursion given in Eq. 3.27. By this approach, only the arc pair a_I is guaranteed to be realized in the precomputed optimal stem pair alignments. So they defined the S items where a_O is not realized as invalid, and skipped cases referring to these items in the core algorithm discussed in 3.4.4 in order to avoid ambiguity. However, they did not give a proof that the optimal global alignment score would not be affected by doing this. Actually, we can compute $S(a_O, a_I)$ in a more intuitive way: we let both a_O and a_I be realized and compute $S(a_O, a_I)$ following the recursion given in Eq. 3.27, then use the results in the core algorithm discussed in 3.4.4.

We call the approach where only a_I is required to be realized *Approach*$_1$, and the approach where only both a_O and a_I are required to be realized *Approach*$_2$. We will prove the following lemma.

Lemma 4.5.6 *For a crossing stem pair $(a_O, a_I) \in ST_{CR}^{all}$, if its outermost arc pair a_O is not realized in the optimal stem pair alignment computed by $Approach_1$, then we can safely remove (a_O, a_I) from ST_{CR}^{all} to avoid unnecessary computation in the core algorithm computing global alignment. This will not affect the optimal global alignment score.*

Proof: For a crossing stem pair $(a_O, a_I) \in ST_{CR}^{all}$, we use $S(a_O, a_I)_{Approach_1}$ to denote its optimal stem pair alignment cost computed by $Approach_1$, and $S(a_O, a_I)_{Approach_2}$ to denote its optimal stem pair alignment cost computed by $Approach_2$.

We use $globalCost((a_O, a_I))$ to represent the optimal global alignment cost with the constraint that the stem pair (a_O, a_I) is realized. We use $subCost(R_1[i_1, i_2], R_2[j_1, j_2])$ to represent the optimal alignment cost between subsequences $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$.

Suppose there is a crossing stem pair $((A, a), (B, b)) \in ST_{CR}^{all}$ whose outermost arc pair (A, a) is not realized in the optimal stem pair alignment computed by $Approach_1$.

It is obvious that $S((A, a), (B, b))_{Approach_1} < S((A, a), (B, b))_{Approach_2}$. The reason is as follows. The computation of the two items only differs in (A, a). In the computation of $S((A, a), (B, b))_{Approach_1}$, when (A, a) is encountered, we take the minimum of all seven cases of the recursion given in Eq. 3.27. Since (A, a) is not realized in the optimal stem pair alignment computed by $Approach_1$, the seventh case realizing (A, a) is not minimal. While in the computation of $S((A, a), (B, b))_{Approach_2}$, when $((A, a)$ is encountered, we simply let (A, a) be realized. Thus $S((A, a), (B, b))_{Approach_1} < S((A, a), (B, b))_{Approach_2}$.

In the optimal stem pair alignment of $((A, a), (B, b))$ produced by $Approach_1$, we suppose that the outermost realized arc pair is (C, c) (see Figure 4.1; this includes these subcases: 1. $C = A$ and $c \neq a$, 2. $C \neq A$ and $c = a$, 3. $C \neq A$ and $c \neq a$).

It is easy to see that $S((A, a), (B, b))_{Approach_1} = subCost(R_1[A^L, C^L -$

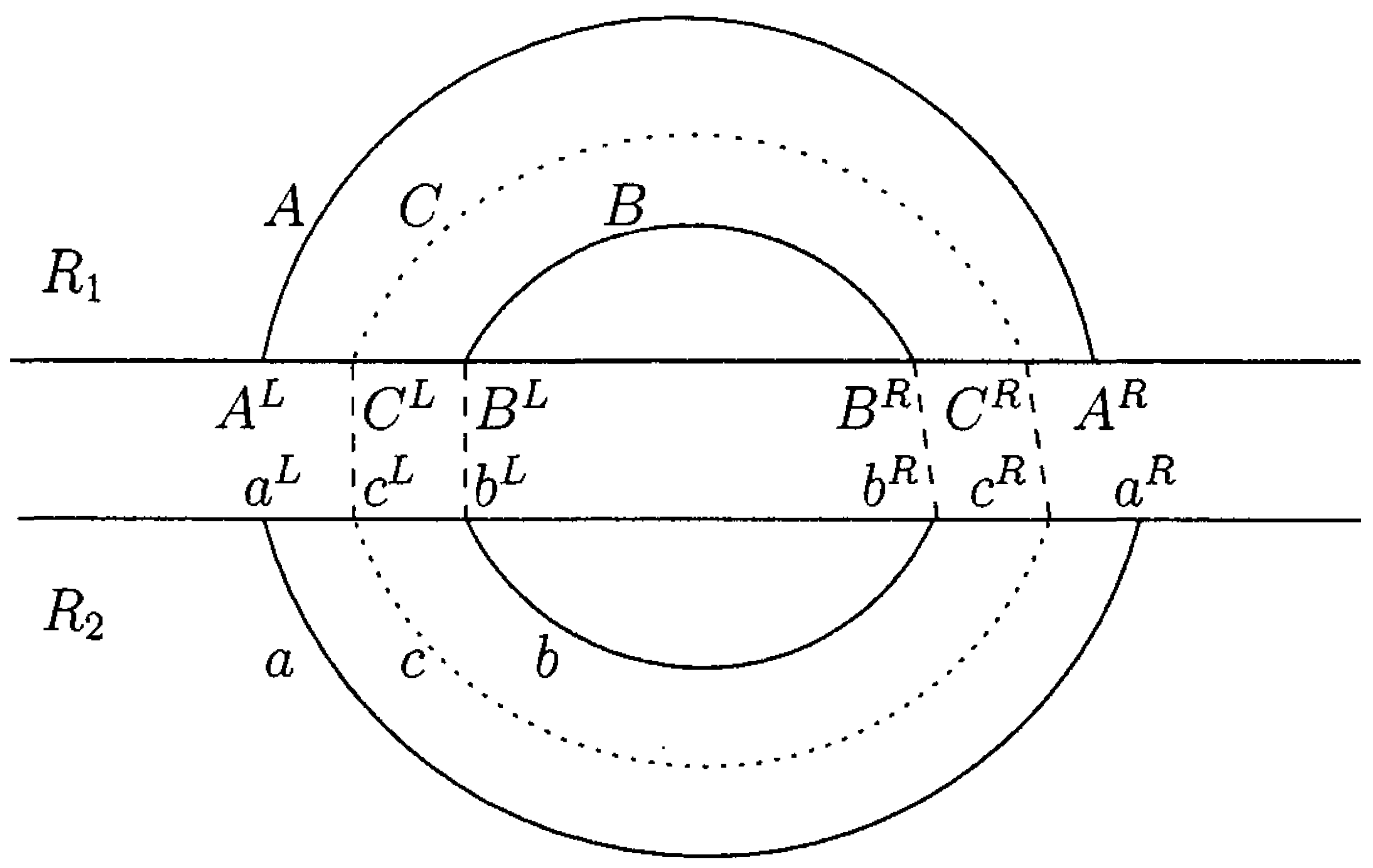


Figure 4.1: Illustration of the proof of Lemma 4.5.6

$1], R_2[a^L, c^L - 1]) + S((C,c),(B,b))_{Approach_1} + subCost(R_1[C^R+1, A^R], R_2[c^R+1, a^R])$[6]. It is obvious that $S((C,c),(B,b))_{Approach_1} = S((C,c),(B,b))_{Approach_2}$.

We have

$$\begin{aligned} & globalCost(((C,c),(B,b))) \\ = \quad & subCost(R_1[1, C^L - 1], R_2[1, c^L - 1]) \\ & + S((C,c),(B,b))_{Approach_2} \\ & + subCost(R_1[B^L + 1, B^R - 1], R_2[b^L + 1, b^R - 1]) \\ & + subCost(R_1[C^R + 1, |R_1|], R_2[c^R + 1, |R_2|]) \end{aligned}$$

Since $subCost(R_1[1, C^L - 1], R_2[1, c^L - 1]) \le subCost(R_1[1, A^L - 1], R_2[1, a^L - 1]) + subCost(R_1[A^L, C^L - 1], R_2[a^L, c^L - 1])$ and $subCost(R_1[C^R + 1, |R_1|], R_2[c^R + 1, |R_2|]) \le subCost(R_1[C^R + 1, A^R], R_2[c^R + 1, a^R]) + subCost(R_1[A^R + 1, |R_1|], R_2[a^R + 1, |R_2|])$,

[6]Recall that the left end of an arc p is denoted as p^L, and the right end is denoted as p^R. If $C = A$ and $c \neq a$, then $subCost(R_1[A^L, C^L - 1], R_2[a^L, c^L - 1])$ will be the cost of insertion of subsequence $R_2[a^L, c^L - 1]$ and $subCost(R_1[C^R + 1, A^R], R_2[c^R + 1, a^R])$ will be the cost of insertion of subsequence $R_2[c^R + 1, a^R]$; if $C \neq A$ and $c = a$, then $subCost(R_1[A^L, C^L - 1], R_2[a^L, c^L - 1])$ will be the cost of deletion of subsequence $R_1[A^L, C^L - 1]$ and $subCost(R_1[C^R + 1, A^R], R_2[c^R + 1, a^R])$ will be the cost of deletion of subsequence $R_1[C^R + 1, A^R]$.

then

$$\begin{aligned}
&\mathit{globalCost}(((C,c),(B,b)))\\
\leq\ & \mathit{subCost}(R_1[1,A^L-1],R_2[1,a^L-1]) + \mathit{subCost}(R_1[A^L,C^L-1],R_2[a^L,c^L-1])\\
&+ S((C,c),(B,b))_{\mathit{Approach}_2}\\
&+ \mathit{subCost}(R_1[B^L+1,B^R-1],R_2[b^L+1,b^R-1])\\
&+ \mathit{subCost}(R_1[C^R+1,A^R],R_2[c^R+1,a^R]) + \mathit{subCost}(R_1[A^R+1,|R_1|],R_2[a^R+1,|R_2|])
\end{aligned}$$

Since $S((A,a),(B,b))_{\mathit{Approach}_1} = \mathit{subCost}(R_1[A^L,C^L-1],R_2[a^L,c^L-1]) + S((C,c),(B,b))_{\mathit{Approach}_1} + \mathit{subCost}(R_1[C^R+1,A^R],R_2[c^R+1,a^R])$ and $S((C,c),(B,b))_{\mathit{Approach}_1} = S((C,c),(B,b))_{\mathit{Approach}_2}$, then

$$\begin{aligned}
&\mathit{globalCost}(((C,c),(B,b)))\\
\leq\ & \mathit{subCost}(R_1[1,A^L-1],R_2[1,a^L-1])\\
&+ S((A,a),(B,b))_{\mathit{Approach}_1}\\
&+ \mathit{subCost}(R_1[B^L+1,B^R-1],R_2[b^L+1,b^R-1])\\
&+ \mathit{subCost}(R_1[A^R+1,|R_1|],R_2[a^R+1,|R_2|])
\end{aligned}$$

Since $S((A,a),(B,b))_{\mathit{Approach}_1} < S((A,a),(B,b))_{\mathit{Approach}_2}$, then

$$\begin{aligned}
&\mathit{globalCost}(((C,c),(B,b)))\\
<\ & \mathit{subCost}(R_1[1,A^L-1],R_2[1,a^L-1])\\
&+ S((A,a),(B,b))_{\mathit{Approach}_2}\\
&+ \mathit{subCost}(R_1[B^L+1,B^R-1],R_2[b^L+1,b^R-1])\\
&+ \mathit{subCost}(R_1[A^R+1,|R_1|],R_2[a^R+1,|R_2|])
\end{aligned}$$

The right part of the above inequality is exactly the same as

$globalCost(((A, a), (B, b)))$, thus we obtain that

$$globalCost(((C, c), (B, b))) < globalCost(((A, a), (B, b)))$$

Therefore, the optimal constrained global alignment realizing $((C, c), (B, b))$ has less cost than the optimal constrained global alignment realizing $((A, a), (B, b))$. Thus realizing the stem pair $((A, a), (B, b))$ will not lead to an optimal global alignment. So we can safely remove $((A, a), (B, b))$ from ST_{CR}^{all} to avoid unnecessary computation in the core algorithm computing global alignment. This will not affect the optimal global alignment score.

Without loss of generality, for a crossing stem pair $(a_O, a_I) \in ST_{CR}^{all}$, if its outermost arc pair a_O is not realized in the optimal stem pair alignment computed by $Approach_1$, then we can safely remove (a_O, a_I) from ST_{CR}^{all} to avoid unnecessary computation in the core algorithm computing global alignment. This will not affect the optimal global alignment score. $\square$

By Lemma 4.5.6, we can compute the alignment cost of crossing stem pairs by $Approach_1$. After the precomputation of all crossing stem pairs is done, we can filter out stem pairs which are not realized. Then we use the filtered crossing stem pairs set which is a subset of ST_{CR}^{all} in the core algorithm computing global alignment. This approach is more efficient than Möhl *et al.*'s approach in which the S items where a_O is not realized are defined as invalid, and cases referring to these items are skipped in the core algorithm. Since large crossing stem pairs set will cause huge space consumption and long runtime (as we will see in Section 4.5.3). Using our approach, we can avoid unnecessary computation and checking in the core algorithm. Although this will not improve the space and time complexities of the algorithm, space and time usage will be significantly improved in practice.

4.5.2.1 Preprocess Crossing Stem Pairs

We now discuss how to preprocess crossing stem pairs. We will show that we only need to compute the alignment of crossing local *m_stem* pairs. Alignment costs of other crossing stem pairs would be byproducts.

We first consider how to compute the optimal alignment cost $S(a_O, a_I)$ of a crossing stem pair $(a_O, a_I) \in ST_{CR}^{all}$ using *Approach*$_1$. The computation of $S(a_O, a_I)$ is based on temporary items $S'(i, i'; j, j'; a_I)$. Let $a_O = ((i_O, p_{r_1}(i_O)), (j_O, p_{r_2}(j_O)))$, $a_I = ((i_I, p_{r_1}(i_I)), (j_I, p_{r_2}(j_I))) \in CR$. Then $S'(i, i'; j, j'; a_I)$ represents the cost of aligning $R_1[i, i_I]$ to $R_2[j, j_I]$ and $R_1[p_{r_1}(i_I), i']$ to $R_2[p_{r_2}(j_I), j']$. $S(((i, i'), (j, j')); a_I)$ is exactly the same as $S'(i, i'; j, j'; a_I)$ if $((i, i'), (j, j'))$ is an arc pair in CR.

We compute $S'(i, i'; j, j'; a_I)$ ($i_O \leq i \leq i_I$, $j_O \leq j \leq j_I$, $p_{r_1}(i_I) \leq i' \leq p_{r_1}(i_O)$, $p_{r_2}(j_I) \leq j' \leq p_{r_2}(j_O)$) using Lemma 4.5.7 and Lemma 4.5.8. The computation starts from a_I and ends when a_O is encountered (see Figure 4.2). The recursion in Lemma 4.5.8 is a modified version of the recursion shown in Eq. 3.27 and allows a more general score scheme. The condition of case (7) of the recursion in Lemma 4.5.8 is a little different from the condition of case (7) of the recursion shown in Eq. 3.27. Both conditions are used to check whether $((i, i'), (j, j'))$ is a crossing arc pair which is in CR. Since an arc pair which is covered by a crossing stem pair must be in CR, we only need to check whether $((i, i'), (j, j'))$ is an arc pair. The condition of case (7) of the recursion in Lemma 4.5.8 cost less checking time.

Lemma 4.5.7

$$S'(i_I, p_{r_1}(i_I); j_I, p_{r_2}(j_I); a_I) = \delta((i_I, p_{r_1}(i_I)), (j_I, p_{r_2}(j_I))) \tag{4.21}$$

Proof: The arc pair $a_I = ((i_I, p_{r_1}(i_I)), (j_I, p_{r_2}(j_I)))$ is realized, that is, the base pair $(R_1[i_I], R_1[p_{r_1}(i_I)])$ is aligned to the base pair $(R_2[j_I], R_2[p_{r_2}(j_I)])$. Thus we have $S'(i_I, p_{r_1}(i_I); j_I, p_{r_2}(j_I); a_I) = \delta((i_I, p_{r_1}(i_I)), (j_I, p_{r_2}(j_I)))$. □

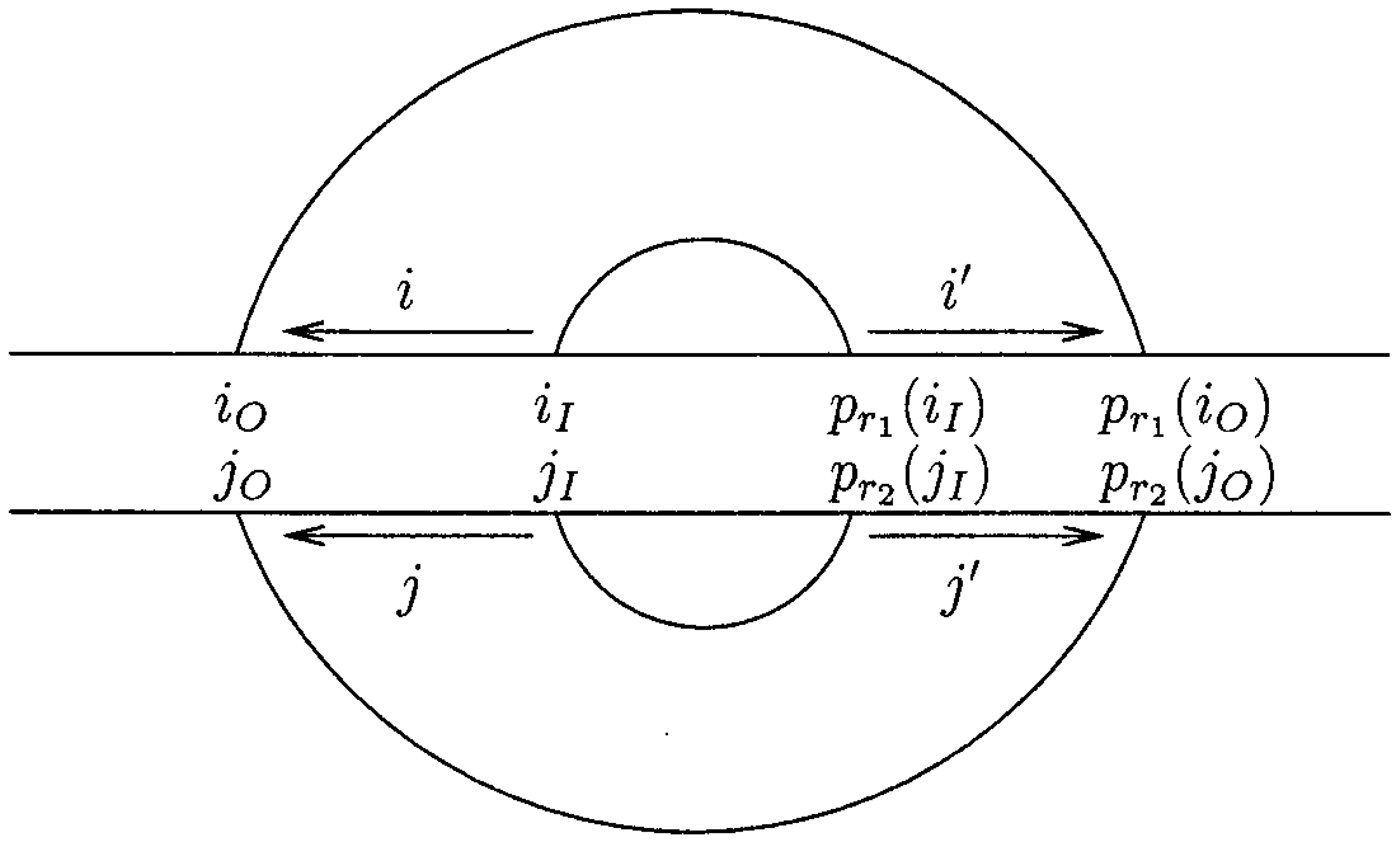


Figure 4.2: A simple illustration of computing the S' item

Lemma 4.5.8 *For $i_O \leq i \leq i_I$, $j_O \leq j \leq j_I$, $p_{r_1}(i_I) \leq i' \leq p_{r_1}(i_O)$, $p_{r_2}(j_I) \leq j' \leq p_{r_2}(j_O)$, and $(i, j, i', j') \neq (i_I, j_I, p_{r_1}(i_I), p_{r_2}(j_I))$*[7]*,*

[7]That is, at least one of the following inequalities $i \neq i_I$, $j \neq j_I$, $i' \neq p_{r_1}(i_I)$, $j' \neq p_{r_2}(j_I)$ holds.

$$S'(i,i';j,j';a_I) =$$

$$\min \begin{cases} \textit{if } i \neq i_I & \\ \quad S'(i+1,i';j,j';a_I) + \gamma(i,0) & \textit{(1)} \\ \textit{if } j \neq j_I & \\ \quad S'(i,i';j+1,j';a_I) + \gamma(0,j) & \textit{(2)} \\ \textit{if } i' \neq p_{r_1}(i_I) & \\ \quad S'(i,i'-1;j,j';a_I) + \gamma(i',0) & \textit{(3)} \\ \textit{if } j' \neq p_{r_2}(j_I) & \\ \quad S'(i,i';j,j'-1;a_I) + \gamma(0,j') & \textit{(4)} \\ \textit{if } i \neq i_I \textit{ and } j \neq j_I & \\ \quad S'(i+1,i';j+1,j';a_I) + \gamma(i,j) & \textit{(5)} \\ \textit{if } i' \neq p_{r_1}(i_I) \textit{ and } j' \neq p_{r_2}(j_I) & \\ \quad S'(i,i'-1;j,j'-1;a_I) + \gamma(i,j) & \textit{(6)} \\ \textit{if } i = p_{r_1}(i') \textit{ and } j = p_{r_2}(j') & \\ \quad S'(i+1,i'-1;j+1,j'-1;a_I) + \delta((i,i'),(j,j')) & \textit{(7)} \end{cases} \tag{4.22}$$

Proof: Consider $R_1[i]$, $R_2[j]$, $R_1[i']$ and $R_2[j']$. There are exactly the following cases.

(1) $R_1[i]$ is aligned to $'-'$. Thus $R_1[i+1,i_I]$ is aligned to $R_2[j,j_I]$ and $R_1[p_{r_1}(i_I),i']$ is aligned to $R_2[p_{r_2}(j_I),j']$. Hence the $S'(i+1,i';j,j';a_I)+\gamma(i,0)$ item. Notice that if $i = i_I$, $i+1$ will be out of the interval $[i_O..i_I]$ and the item $S'(i+1,i';j,j';a_I)$ is invalid. Thus for this situation, we need to skip case (1). So we add the condition $i \neq i_I$ to case (1).

(2) $R_2[j]$ is aligned to $'-'$. Similar to case (1). We need to add the condition $j \neq j_I$ to case (2).

(3) $R_1[i']$ is aligned to $'-'$. Similar to case (1). We need to add the condition

$i' \neq p_{r_1}(i_I)$ to case (3).

(4) $R_2[j']$ is aligned to '$-$'. Similar to case (1). We need to add the condition $j' \neq p_{r_2}(j_I)$ to case (4).

(5) $R_1[i]$ is aligned to $R_2[j]$, but no arc pair involving $(R_1[i], R_2[j])$ is realized. Thus $R_1[i+1, i_I]$ is aligned to $R_2[j+1, j_I]$ and $R_1[p_{r_1}(i_I), i']$ is aligned to $R_2[p_{r_2}(j_I), j']$. Hence the $S'(i+1, i'; j+1, j'; a_I) + \gamma(i,j)$ item. Similar to the reason in case (1), we add the condition $i \neq i_I$ and $j \neq j_I$ to case (5).

(6) $R_1[i']$ is aligned to $R_2[j']$, but no arc pair involving $(R_1[i'], R_2[j'])$ is realized. Similar to case (5). We need to add the condition $i' \neq p_{r_1}(i_I)$ and $j' \neq p_{r_2}(j_I)$ to case (6).

(7) $R_1[i]$ is aligned to $R_2[j]$, and $((i,i'),(j,j'))$ is a crossing arc pair and it is realized. Then the optimal alignment is partitioned into two parts: 1. the alignment of the base pair $(R_1[p_{r_1}(i)], R_1[i])$ to the base pair $(R_2[p_{r_2}(j)], R_2[j])$, and 2. the remaining part. Hence the $S'(i+1, i'-1; j+1, j'-1; a_I) + \delta((p_{r_1}(i), i), (p_{r_2}(j), j))$ item.

Therefore we take the minimum of all the cases and get the above recursion. □

By the meaning of the S, S' items, and Lemma 4.5.6 and Lemma 4.5.8, we only need to compute the alignment of crossing inner local *m_stem* pairs. Alignment costs of other crossing stem pairs would be byproducts. The details are as follows. During the computation of $S'(i, i'; j, j'; a_I)$ for a crossing inner local *m_stem* pair (a_O, a_I), when current position $(i, i'; j, j')$ corresponds to an arc pair $((i,i'),(j,j')) \in CR$ and this arc pair is realized (i.e. case(7) of Eq. 4.22 is the minimum at this position), we keep a record of the stem pair $(((i,i'),(j,j')); a_I)$ with current alignment score as its alignment cost[8]. Thus when the computation is done, we will have all realized stem pairs which have the same innermost arc pair as (a_O, a_I) with associated alignment costs.

For each crossing inner local *m_stem* pair (a_O, a_I) where a_O =

[8]We also record the stem pair $(a_I; a_I)$ which is actually an arc pair a_I, since we also consider a single arc as a stem and $(a_I; a_I)$ is obviously realized.

$((i_O, p_{r_1}(i_O)), (j_O, p_{r_2}(j_O)))$ and $a_I = ((i_I, p_{r_1}(i_I)), (j_I, p_{r_2}(j_I)))$, we can compute its alignment cost $S(a_O, a_I) = S'(i_O, p_{r_1}(i_O); j_O, p_{r_2}(j_O); a_I)$ using Lemma 4.5.7 and Lemma 4.5.8. Algorithm 4.1 shows how to compute $S'(i_O, p_{r_1}(i_O); j_O, p_{r_2}(j_O); a_I)$.

Algorithm 4.1 *Inner-Local-Max-Stem-Pair-Align*$((a_O, a_I))$

Input: Inner local *m_stem* pair (a_O, a_I) where $a_O = ((i_O, p_{r_1}(i_O)), (j_O, p_{r_2}(j_O)))$ and $a_I = ((i_I, p_{r_1}(i_I)), (j_I, p_{r_2}(j_I)))$.

Output: Alignment score matrix $T(i_O..i_I; j_O..j_I; p_{r_1}(i_I)..p_{r_1}(i_O); p_{r_2}(j_I)..p_{r_2}(j_O))$, list L of crossing stem pairs, and array *scoreL* containing the corresponding alignment costs of these stem pairs.

```
1: compute $S'(i_I, p_{r_1}(i_I); j_I, p_{r_2}(j_I); a_I)$ as in Lemmas 4.5.7, append the stem pair $(a_I; a_I)$ to $L$ and append $S'(i_I, p_{r_1}(i_I); j_I, p_{r_2}(j_I); a_I)$ to scoreL
2: for $i \leftarrow i_I$ downto $i_O$ do
3:   for $j \leftarrow j_I$ downto $j_O$ do
4:     for $i' \leftarrow p_{r_1}(i_I)$ to $p_{r_1}(i_O)$ do
5:       for $j' \leftarrow p_{r_2}(j_I)$ to $p_{r_2}(j_O)$ do
6:         if $(i, j, i', j') \neq (i_I, j_I, p_{r_1}(i_I), p_{r_2}(j_I))$ then
7:           compute $S'(i, i'; j, j'; a_I)$ as in Lemma 4.5.8, when the condition of case (7) of the recursion is satisfied and case (7) is the minimum among all seven cases, append the stem pair $(((i, i'), (j, j')); a_I)$ to $L$ and append $S'(i, i'; j, j'; a_I)$ to scoreL
8:         end if
9:       end for
10:     end for
11:   end for
12: end for
13: return $(L, scoreL)$
```

Crossing inner local *m_stem* pairs can be easily generated from ST_{CR}^{MAX}. We generate crossing inner local *m_stem* pairs in the following way.

For a crossing *m_stem* pair $(a_{O_{MAX}}, a_{I_{MAX}})$, where $a_{O_{MAX}} = (p_{O_1}, p_{O_2})$ and $a_{I_{MAX}} = (p_{I_1}, p_{I2})$, we first select an arc pair $a_I = (p_1, p_2)$ covered by $(a_{O_{MAX}}, a_{I_{MAX}})$. If $p_1 = p_{O_1}$ or $p_2 = p_{O_2}$, then the stem pair $(a_I, a_I) = ((p_1, p_2), (p_1, p_2))$ is inner local maximal[9] (see cases (1) and (2) of Figure 4.3); else the stem pair $(a_{O_{MAX}}, a_I) = ((p_{O_1}, p_{O_2}), (p_1, p_2))$ is inner local maximal (see case (3) of Figure 4.3). We iterate

[9]$(a_I; a_I)$ is actually an arc pair a_I. Since we also consider a single arc as a stem, $(a_I; a_I)$ can be considered as a stem pair.

over all possible alternatives of a_I, then can get all crossing inner local *m_stem* pairs covered by the *m_stem* pair $(a_{O_{MAX}}, a_{I_{MAX}})$. For each *m_stem* pair, we do the above step. Then we can get the set of crossing inner local *m_stem* pairs.

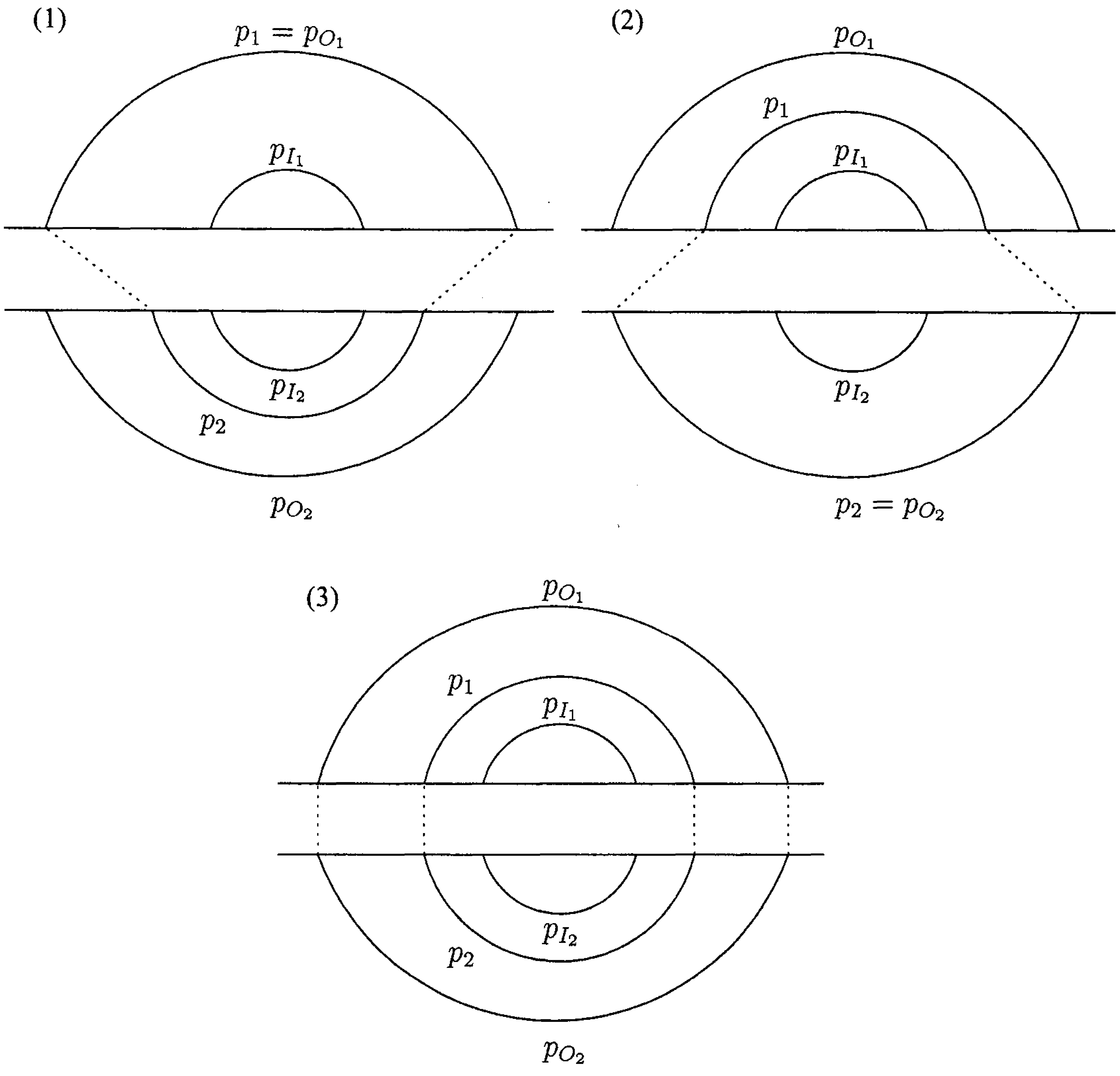


Figure 4.3: A simple illustration of inner local *m_stem* pair

By Algorithm 4.1, we do not need to generate ST_{CR}^{all} from ST_{CR}^{MAX} directly. After the precomputation of all crossing inner local *m_stem* pairs is done, we will get a set of filtered crossing stem pairs which are realized. Many crossing stem pairs in ST_{CR}^{all} will not be realized in practice, thus this filtered set is usually much smaller than the original ST_{CR}^{all}. In the remaining part of this thesis, we will simply use ST_{CR} to denote this filtered crossing stem pairs set which is a subset of ST_{CR}^{all}. We use this set in the core algorithm computing global alignment to avoid unnecessary computation.

For the set ST_{CR} of crossing stem pairs produced by the above method, we compute a sorted list of ST_{CR} by $5'$ ends of two RNAs. For stem pairs $(a_O, a_I) \in ST_{CR}$, we sort (a_O, a_I) by the left end of the outermost arc pair $\nwarrow a_O = (i, j)$ according to the partial order $\lessdot$.

Algorithm 4.2 shows the whole method to preprocess the crossing stem pairs.

Algorithm 4.2 *Preprocess-Crossing-Stem-Pairs*(ST_{CR}^{MAX})

Input: Sorted (by $5'$ ends) list ST_{CR}^{MAX} of crossing *m_stem* pairs.

Output: Sorted (by $5'$ ends) list ST_{CR} of crossing stem pairs which has been filtered, and array $scoreST_{CR}$ containing the corresponding alignment costs of these stem pairs.

```
1: for $k \leftarrow 1$ to $|ST_{CR}^{MAX}|$ do
2:   let $ST_{CR}^{MAX}[k] = ((p_{O_1}, p_{O_2}), (p_{I_1}, p_{I_2}))$
3:   for $i \leftarrow p_{O_1}^L$ to $p_{I_1}^L$ do
4:     if $i \neq p_{r_1}(i)$ then
5:       for $j \leftarrow p_{O_2}^L$ to $p_{I_2}^L$ do
6:         if $j \neq p_{r_2}(j)$ then
7:           if $i = p_{O_1}^L$ or $j = p_{O_2}^L$ then
8:             $a_I \leftarrow ((i, p_{r_1}(i)), (j, p_{r_2}(j)))$
9:             $(L, scoreL) \leftarrow$ Inner-Local-Max-Stem-Pair-Align$((a_I, a_I))$ // Algorithm 4.1
10:            append $L$ to $ST_{CR}$
11:            append $scoreL$ to $scoreST_{CR}$
12:          else
13:            $a_I \leftarrow ((i, p_{r_1}(i)), (j, p_{r_2}(j)))$
14:            $a_O \leftarrow (p_{O_1}, p_{O_2})$
15:            $(L, scoreL) \leftarrow$ Inner-Local-Max-Stem-Pair-Align$((a_O, a_I))$
16:            append $L$ to $ST_{CR}$
17:            append $scoreL$ to $scoreST_{CR}$
18:          end if
19:        end if
20:      end for
21:    end if
22:  end for
23: end for
24: sort $ST_{CR}$ by $5'$ ends of two RNAs; the order of elements of $scoreST_{CR}$ also
    changes with $ST_{CR}$ such that $scoreST_{CR}[i]$ is the alignment score of $ST_{CR}[i]$
```

Similar to Algorithm 4.1 and Algorithm 4.2, we can generate all crossing outer local *m_stem* pairs and compute their alignment. We will get another filtered crossing stem pairs set ST'_{CR} and its sorted list.

We can compute the intersection of ST_{CR} produced by crossing inner local *m_stem* pairs and ST'_{CR} produced by crossing outer local *m_stem* pairs, and use this intersection as the set of crossing stem pairs in the core algorithm. By doing this, we can filter out more crossing stem pairs and the core algorithm will be faster.

4.5.3 Formula for Computing Optimal Alignment

We use a bottom up dynamic programming algorithm to find the optimal alignment between R_1 and R_2. We consider the smaller substructures first and eventually consider the whole structures R_1 and R_2.

We first consider how to compute the optimal alignment between $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$.

With the techniques discussed in the previous section, we can determine the alignment score of all crossing stem pairs in preprocessing. We update Lemma 4.4.1 to Lemma 4.4.3 to the stem pair version. We use $A(i_1, i; j_1, j|M)$ to represent the optimal alignment cost between $R_1[i_1, i]$ and $R_2[j_1, j]$ where $M \subseteq ST_{CR}$ is a set of its proper open stem pairs[10]. Lemma 4.5.9 to Lemma 4.5.11 correspond to Lemma 4.4.1 to Lemma 4.4.3, respectively. The recursion in Lemma 4.5.11 is a modified version of the recursion shown in Eq. 3.30 and allows a more general score scheme.

Lemma 4.5.9

$$A(\emptyset; \emptyset|\emptyset) = 0 \tag{4.23}$$

Proof: Similar to Lemma 4.4.1. □

[10]The definition of proper open stem pairs set is in Section 4.1.

Lemma 4.5.10 *For $i_1 \leq i \leq i_2$,*

$$A(i_1, i; \emptyset|\emptyset) = A(i_1, i-1; \emptyset|\emptyset) + \gamma(i, 0) \tag{4.24}$$

For $j_1 \leq j \leq j_2$,

$$A(\emptyset; j_1, j|\emptyset) = A(\emptyset; j_1, j-1|\emptyset) + \gamma(0, j) \tag{4.25}$$

Proof: Similar to Lemma 4.4.2. □

Lemma 4.5.11 *For $i_1 \leq i \leq i_2$ and $j_1 \leq j \leq j_2$,*

$$A(i_1, i; j_1, j|M) =$$

$$\min \begin{cases}
\textit{skip (1) if there exists some stem pair } (a_O, a_I) \in M \textit{ with} \\
\quad \nwarrow a_I = (i, j') \textit{ or } \searrow a_O = (i, j') \textit{ where } j_1 \le j' \le j \\
\quad A(i_1, i-1; j_1, j|M) + \gamma(i, 0) \\
\textit{skip (2) if there exists some stem pair } (a_O, a_I) \in M \textit{ with} \\
\quad \nwarrow a_I = (i', j) \textit{ or } \searrow a_O = (i', j) \textit{ where } i_1 \le i' \le i \\
\quad A(i_1, i; j_1, j-1|M) + \gamma(0, j) \\
\textit{skip (3) if there exists some stem pair } (a_O, a_I) \in M \textit{ with} \\
\quad \nwarrow a_I = (i, j') \textit{ or } \searrow a_O = (i, j') \textit{ where } j_1 \le j' \le j, \textit{ or} \\
\quad \nwarrow a_I = (i', j) \textit{ or } \searrow a_O = (i', j) \textit{ where } i_1 \le i' \le i \\
\quad A(i_1, i-1; j_1, j-1|M) + \gamma(i, j) \\
\textit{if } (i_1, j_1) \preceq (p_{r_1}(i), p_{r_2}(j)) \prec (i, j) \textit{ and } ((p_{r_1}(i), i), (p_{r_2}(j), j)) \in NC, \\
\quad \textit{and } ((p_{r_1}(i), i), (p_{r_2}(j), j)) \textit{ is compatible with } M \\
\quad \min\limits_{(M_1, M_2)} \left\{ \begin{array}{l|l}
 & M_1, M_2 \subseteq ST_{CR}, \textit{ where} \\
A(i_1, p_{r_1}(i)-1; j_1, p_{r_2}(j)-1|M_1) & M_{share} = M_1 \cap M_2, \\
+ A(p_{r_1}(i)+1, i-1; p_{r_2}(j)+1, j-1|M_2) & M = (M_1 \cup M_2) - M_{share} \\
+ \delta((p_{r_1}(i), i), (p_{r_2}(j), j)) & \textit{and } M_{share} \textit{ is} \\
 & \textit{compatible with } M
\end{array} \right\} \\
\textit{if there exists some stem pair } (a_O, a_I) \in M \textit{ with} \\
\quad \nwarrow a_O = (i', j') \wedge \nwarrow a_I = (i, j) \textit{ or } \searrow a_I = (i', j') \wedge \searrow a_O = (i, j) \\
\quad A(i_1, i'-1; j_1, j'-1|M - \{(a_O, a_I)\}) + S(a_O, a_I)/2 \\
\quad \min\limits_{(i', j')} \left\{ \begin{array}{l|l}
 & (a_O, a_I) \in ST_{CR}, \textit{ where} \\
 & \searrow a_O = (i, j) \textit{ and} \\
A(i_1, i'-1; j_1, j'-1|M \cup \{(a_O, a_I)\}) & \searrow a_I = (i', j'), \textit{ and} \\
+ S(a_O, a_I)/2 & (a_O, a_I) \textit{ is compatible} \\
 & \textit{with } M
\end{array} \right\}
\end{cases} \tag{4.26}$$

Proof: Consider $R_1[i]$ and $R_2[j]$. There are exactly the following cases.

(1) $R_1[i]$ is aligned to '$-$'. Thus $R_1[i_1, i-1]$ is aligned to $R_2[j_1, j]$ and the proper open stem pairs set is still M. Hence the $A(i_1, i-1; j_1, j|M) + \gamma(i, 0)$ item. Notice that if there exists some stem pair $(a_O, a_I) \in M$ with $\nwarrow a_I = (i, j')$ where $j_1 \leq j' \leq j$, then $R_1[i]$ must be aligned to $R_2[j']$ since a_I is the innermost arc pair of an open stem pair and realized in the alignment; if there exists some stem pair $(a_O, a_I) \in M$ with $\searrow a_O = (i, j')$ where $j_1 \leq j' \leq j$, then $R_1[i]$ must be aligned to $R_2[j']$ for similar reason. Thus for these situations, we need to skip case (1).

(2) $R_2[j']$ is aligned to '$-$'. Similar to case (1). Notice that if there exists some stem pair $(a_O, a_I) \in M$ with $\nwarrow a_I = (i', j)$ or $\searrow a_O = (i', j)$ where $i_1 \leq i' \leq i$, we need to skip case (2) due to similar reason in case (1).

(3) $R_1[i]$ is aligned to $R_2[j]$, but no arc pair involving $(R_1[i], R_2[j])$ is realized. Thus $R_1[i_1, i-1]$ is aligned to $R_2[j_1, j-1]$ and the proper open stem pairs set is still M. Hence the $A(i_1, i-1; j_1, j-1|M) + \gamma(i, j)$ item. Notice that if there exists some stem pair $(a_O, a_I) \in M$ with $\nwarrow a_I = (i, j')$ or $\searrow a_O = (i, j')$ where $j_1 \leq j' \leq j$, or $\nwarrow a_I = (i', j)$ or $\searrow a_O = (i', j)$ where $i_1 \leq i' \leq i$, we need to skip case (3) due to similar reason in case (1).

(4) $R_1[i]$ is aligned to $R_2[j]$, and the arc pair $((p_{r_1}(i), i), (p_{r_2}(j), j))$ is realized. This requires that $((p_{r_1}(i), i), (p_{r_2}(j), j))$ is compatible with M. Then the optimal alignment between $R_1[i_1, i]$ and $R_2[j_1, j]$ is partitioned into three parts: 1. the optimal alignment between $R_1[i_1, p_{r_1}(i) - 1]$ and $R_2[j_1, p_{r_2}(j) - 1]$, 2. the optimal alignment between $R_1[p_{r_1}(i) + 1, i - 1]$ and $R_2[p_{r_2}(j) + 1, j - 1]$, and 3. the alignment of $(R_1[p_{r_1}(i)], R_1[i])$ to $(R_2[p_{r_2}(j)], R_2[j])$. For part 1 and part 2, we denote their corresponding proper open stem pairs sets by M_1 and M_2, respectively. Hence the $A(i_1, p_{r_1}(i) - 1; j_1, p_{r_2}(j) - 1|M_1) + A(p_{r_1}(i) + 1, i - 1; p_{r_2}(j) + 1, j - 1|M_2) + \delta((p_{r_1}(i), i), (p_{r_2}(j), j))$ item. M_1 and M_2 may contain open stem pairs which are not contained in M. We use M_{share} to denote the set of stem pairs that are shared between part 1 and part 2, i.e. $M_{share} = M_1 \cap M_2$. M, M_1, M_2, M_{share} satisfy that $M = (M_1 \cup M_2) - M_{share}$ and M_{share} is compatible with M. We need to minimize

over all possible alternatives.

(5) We have two subcases. (a) The innermost arc pair of a proper open stem pair $(a_O, a_I) \in M$ has left end in (i, j). (a_O, a_I) is uniquely determined. Assume the left end of the outermost arc pair a_O of (a_O, a_I) is (i', j'). Then the optimal alignment between $R_1[i_1, i]$ and $R_2[j_1, j]$ is partitioned into two parts: 1. the optimal alignment between $R_1[i_1, i'-1]$ and $R_2[j_1, j'-1]$, 2. the left part of the alignment of stem pair (a_O, a_I). For part 1, (a_O, a_I) is no more open, thus its proper open stem pairs set is $M - \{(a_O, a_I)\}$, and the alignment cost is $A(i_1, i'-1; j_1, j'-1|M - \{(a_O, a_I)\})$. For part 2, the cost is half of the cost of aligning stem pair (a_O, a_I), that is $S(a_O, a_I)/2$. (b) The outermost arc pair of a stem pair $(a_O, a_I) \in M$ has right end in (i, j). It is similar to subcase (a).

(6) The outermost arc pair of a stem pair $(a_O, a_I) \in ST_{CR}$ has right end in (i, j), (a_O, a_I) is compatible with M, and (a_O, a_I) is not open for the alignment between $R_1[i_1, i]$ and $R_2[j_1, j]$. Assume the right end of the innermost arc pair a_I of (a_O, a_I) is (i', j'). Then the optimal alignment between $R_1[i_1, i]$ and $R_2[j_1, j]$ is partitioned into two parts: 1. the optimal alignment between $R_1[i_1, i'-1]$ and $R_2[j_1, j'-1]$, 2. the right part of the alignment of stem pair (a_O, a_I). For part 1, (a_O, a_I) is now open, thus its proper open stem pairs set is $M \cup \{(a_O, a_I)\}$, and the alignment cost is $A(i_1, i'-1; j_1, j'-1|M \cup \{(a_O, a_I)\})$. For part 2, the cost is half of the cost of aligning stem pair (a_O, a_I), that is $S(a_O, a_I)/2$. Notice that a_I may have multiple instances. We need to minimize over all possible alternatives.

Therefore we take the minimum of all the cases and get the above recursion. □

From case (4) of Lemma 4.5.11, during the computation of alignment between $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$, when the position of (i, j) corresponds to the right end of a non-crossing arc pair $((p_{r_1}(i), i), (p_{r_2}(j), j)) \in NC$, we need to use the optimal subalignment score $A(p_{r_1}(i) + 1, i - 1; p_{r_2}(j) + 1, j - 1|M_2)$ of the segment $(p_{r_1}(i) + 1, i - 1; p_{r_2}(j) + 1, j - 1)$ enclosed by the arc pair $((p_{r_1}(i), i), (p_{r_2}(j), j))$. Thus we need to compute $A(p_{r_1}(i)+1, i-1; p_{r_2}(j)+1, j-1|M_2)$ before we compute alignment between $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$. Obviously, for $A(p_{r_1}(i)+1, i-1; p_{r_2}(j)+1, j-1|M_2)$, there may be multiple instances of M_2. We need to know all possible alternatives and enumerate them. Thus we need to precompute a list of all possible proper open stem pairs sets for the segment $(p_{r_1}(i)+1, i-1; p_{r_2}(j)+1, j-1)$. Assume this list is M_{list}. Then we need to compute all possible $A(p_{r_1}(i)+1, i-1; p_{r_2}(j)+1, j-1|M_2)$ where $M_2 \in M_{list}$. We can save these subalignment scores in an array for later use (for case (4) of Lemma 4.5.11).

Here we distinguish two kinds of subalignments of the segment $(p_{r_1}(i)+1, i-1; p_{r_2}(j)+1, j-1)$: one has no open stem pairs, i.e. $M_2 = \emptyset$; and the other one has open stem pairs, i.e. $M_2 \neq \emptyset$. The computation of these two kinds of subalignments is a little different. We will discuss methods to compute these two kinds of subalignments in Section 4.5.5 and Section 4.5.6 separately.

We first consider how to precompute all possible proper open stem pairs sets for all segments enclosed by non-crossing arc pairs in NC.

4.5.4 Computing All Possible Proper Open Stem Pairs Sets

We now consider how to generate all possible proper open stem pairs sets for segments enclosed by non-crossing arc pairs in NC.

For a segment $(i_1, i_2; j_1, j_2)$ enclosed by an arc pair $a \in NC$ which is covered by the non-crossing *m_stem* pair $((p_{O_1}, p_{O_2}), (p_{I_1}, p_{I_2})) \in ST_{NC}^{MAX}$, the list of possible proper open stem pairs sets of the segment $(i_1, i_2; j_1, j_2)$ is exactly the same as the list of possible proper open stem pairs sets of the segment $(p_{I_1}^L + 1, p_{I_1}^R - 1; p_{I_2}^L + 1, p_{I_2}^R - 1)$. So we only need to compute lists of possible proper open stem pairs sets for segments enclosed by the innermost arc pairs of non-crossing *m_stem* pairs in ST_{NC}^{MAX}. These lists can be easily generated from the crossing stem pairs set ST_{CR}.

Now we can discuss how to generate a list of all possible proper open stem pairs sets for the segment $(i_1, i_2; j_1, j_2)$ which is enclosed by the innermost arc pair of a non-crossing *m_stem* pair in ST_{NC}^{MAX}. First, we put each stem pair in ST_{CR} which has one end inside $(i_1, i_2; j_1, j_2)$ and the other end outside $(i_1, i_2; j_1, j_2)$ into a set M_{set}

(see Figure 4.4). Formally, if a stem pair $(a_O, a_I) \in ST_{CR}$ satisfies that $\nwarrow a_I \prec (i_1, j_1) \preceq \searrow a_I$ and $\searrow a_O \preceq (i_2, j_2)$, or $(i_1, j_1) \preceq \nwarrow a_O$ and $\nwarrow a_I \preceq (i_2, j_2) \prec \searrow a_I$, we put it into M_{set}. Then we generate all subsets of M_{set} such that in each set, the stem pairs are compatible with each other. The list of these subsets is exactly what we want. Notice that $\emptyset$ is also included in the list.

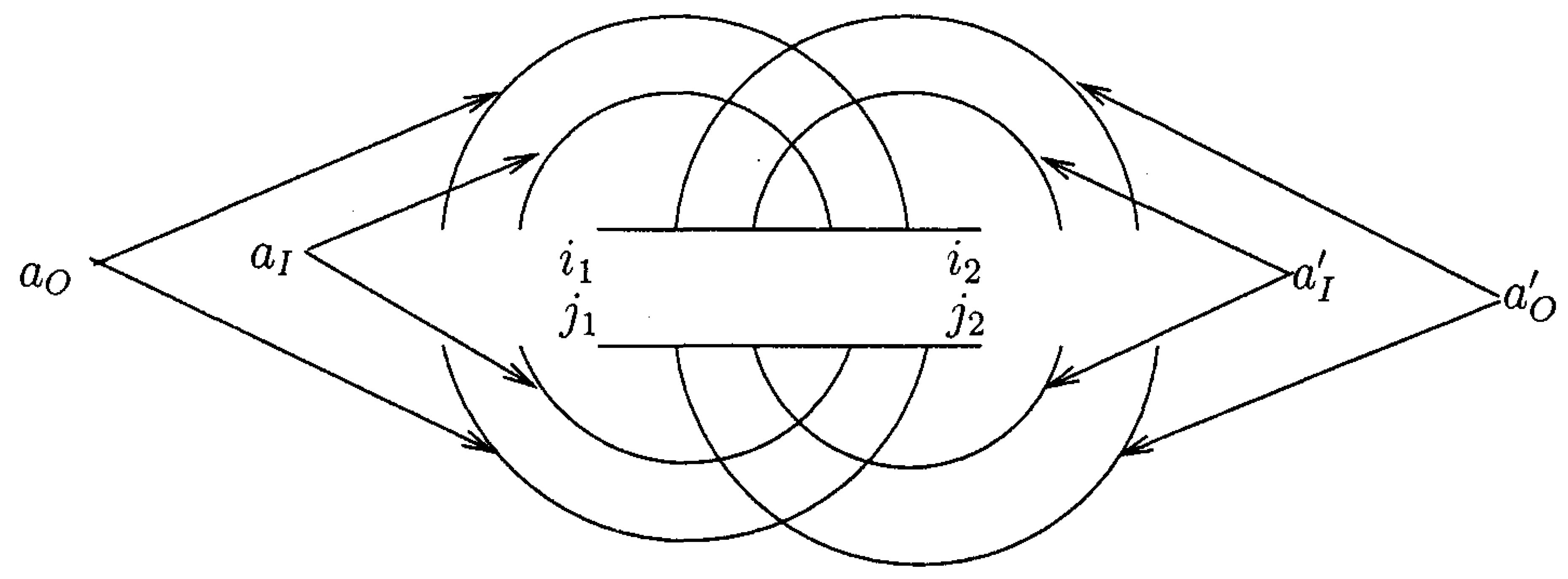


Figure 4.4: A simple illustration of open stem pairs of a segment $(i_1, i_2; j_1, j_2)$

The algorithm for generating lists of all possible proper open stem pairs sets for segments enclosed by the innermost arc pairs of non-crossing *m_stem* pairs in ST_{NC}^{MAX} is shown in Algorithm 4.3[11]. We save the results according to non-crossing *m_stem* pairs for convenient reference. We can precompute an array which maps each non-crossing arc pair $a \in NC$ to the non-crossing *m_stem* pair in ST_{NC}^{MAX} which covers a. Then when we compute the alignment of $(i_1, i_2; j_1, j_2)$ enclosed by a non-crossing arc pair $a \in NC$, we can easily find which non-crossing *m_stem* pair covers a and obtain the corresponding list of all possible proper open stem pairs sets.

4.5.5 Algorithm for Computing Optimal Subalignment without Open Stem pairs

In this section, we discuss how to compute optimal alignment between $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$ without open stem pairs, i.e. $A(i_1, i_2; j_1, j_2|\emptyset)$. We will show that we need

[11]The procedure *Append-Stem-Pair(b, C)* in Algorithm 4.3 gets a stem pair b and a list C of stem pairs as input, and returns a list D of stem pairs containing all elements of C and b. b is at the end of D. It is not central to the thesis, thus we do not include the details of this procedure.

Algorithm 4.3 *Gen-OSP-for-NC-Seg*(ST_{NC}^{MAX}, ST_{CR})

Input: Sorted (by 3′ ends) list ST_{NC}^{MAX} of non-crossing *m_stem* pairs, and sorted (by 5′ ends) list ST_{CR} of crossing stem pairs.

Output: Jagged array $OSP[\ ][\ ]$ where $OSP[k]$ is list of all possible proper open stem pairs lists of the segment enclosed by the innermost arc pair of $ST_{NC}^{MAX}[k]$ and $OSP[k][i]$ is the i-th proper open stem pairs list in $OSP[k]$.

```
1: for $k \leftarrow 1$ to $|ST_{NC}^{MAX}|$ do
2:   let $ST_{NC}^{MAX}[k] = ((p_{O_1}, p_{O_2}), (p_{I_1}, p_{I2}))$, and $i_1 = p_{I_1}^L + 1$, $i_2 = p_{I_1}^R - 1$, $j_1 = p_{I_2}^L + 1$,
     $j_2 = p_{I_2}^R - 1$
3:   for $i \leftarrow 1$ to $|ST_{CR}|$ do
4:     if $ST_{CR}[i]$ has one end inside $(i_1, i_2; j_1, j_2)$ and the other end outside
       $(i_1, i_2; j_1, j_2)$ then
5:       append $ST_{CR}[i]$ to $M_{set}$
6:     end if
7:   end for
8:   append $\emptyset$ to $OSP[k]$
9:   for $i \leftarrow 1$ to $|OSP[k]|$ do
10:    for $j \leftarrow 1$ to $|M_{set}|$ do
11:      if $M_{set}[j]$ is compatible with $OSP[k][i]$ then
12:        append Append-Stem-Pair($M_{set}[j]$, $OSP[k][i]$) to $OSP[k]$
13:      end if
14:    end for
15:  end for
16: end for
```

to maintain multiple matrices to hold alignment scores. These matrices are generated as needed and are indexed by the generated order. Each matrix has a corresponding proper open stem pairs set. These matrices may have different size. We will discuss how to determine the size of these matrices. At a given position (i, j), only some matrices exist. Thus we need to know which matrices exist at (i, j). We will address these problems in the remaining part of this section.

4.5.5.1 Modifying Conditions of Lemma 4.5.11

Since we require that the optimal subalignment has no open stem pairs, crossing stem pairs which have one end inside $(i_1, i_2; j_1, j_2)$ and the other end outside $(i_1, i_2; j_1, j_2)$ will not be realized. Thus we need to modify Lemma 4.5.11 a little to compute $A(i_1, i_2; j_1, j_2|\emptyset)$. The changes are as follows.

(a) We change the condition in case (1) from:

- *skip (1) if there exists some stem pair* $(a_O, a_I) \in M$ *with* $\nwarrow a_I = (i, j')$ *or* $\searrow a_O = (i, j')$ *where* $j_1 \le j' \le j$

to:

- *skip (1) if there exists some stem pair* $(a_O, a_I) \in M$ *with* $(i_1, j_1) \preceq \nwarrow a_O \prec \searrow a_O \preceq (i_2, j_2)$ *and* $\nwarrow a_I = (i, j')$ *where* $j_1 \le j' \le j$.

The reason is as follows. As we have said, crossing stem pairs which have one end inside $(i_1, i_2; j_1, j_2)$ and the other end outside $(i_1, i_2; j_1, j_2)$ will not be realized in the optimal subalignment of $(i_1, i_2; j_1, j_2)$. Thus they cannot be in the set of proper open stems M. The stem pairs $(a_O, a_I) \in M$ must satisfy that $(i_1, j_1) \preceq \nwarrow a_O \prec \searrow a_O \preceq (i_2, j_2)$. If there exists some stem pair $(a_O, a_I) \in M$ with $\nwarrow a_I = (i, j')$ where $j_1 \le j' \le j$, then $R_1[i]$ must be aligned to $R_2[j']$ since a_I is the innermost arc pair of an open stem pair and realized in the alignment, thus we need to skip case (1).

(b) We change the condition in case (2) from:

- *skip (2) if there exists some stem pair* $(a_O, a_I) \in M$ *with* $\nwarrow a_I = (i', j)$ *or* $\searrow a_O = (i', j)$ *where* $i_1 \leq i' \leq i$

to:

- *skip (2) if there exists some stem pair* $(a_O, a_I) \in M$ *with* $(i_1, j_1) \preceq \nwarrow a_O \prec \searrow a_O \preceq (i_2, j_2)$ *and* $\nwarrow a_I = (i', j)$ *where* $i_1 \leq i' \leq i$.

The reason is similar to the reason for changing the condition in case (1).

(c) We change the condition in case (3) from:

- *skip (3) if there exists some stem pair* $(a_O, a_I) \in M$ *with* $\nwarrow a_I = (i, j')$ *or* $\searrow a_O = (i, j')$ *where* $j_1 \leq j' \leq j$, *or* $\nwarrow a_I = (i', j)$ *or* $\searrow a_O = (i', j)$ *where* $i_1 \leq i' \leq i$

to:

- *skip (3) if there exists some stem pair* $(a_O, a_I) \in M$ *with* $(i_1, j_1) \preceq \nwarrow a_O \prec \searrow a_O \preceq (i_2, j_2)$, *and* $\nwarrow a_I = (i, j')$ *where* $j_1 \leq j' \leq j$ *or* $\nwarrow a_I = (i', j)$ *where* $i_1 \leq i' \leq i$.

The reason is similar to the reason for changing the condition in case (1).

(d) We change the condition in case (5) from:

- *if there exists some stem pair* $(a_O, a_I) \in M$ *with* $\nwarrow a_O = (i', j') \wedge \nwarrow a_I = (i, j)$ *or* $\searrow a_I = (i', j') \wedge \searrow a_O = (i, j)$

to:

- *if there exists some stem pair* $(a_O, a_I) \in M$ *with* $(i_1, j_1) \preceq \nwarrow a_O \prec \searrow a_O \preceq (i_2, j_2)$ *and* $\nwarrow a_O = (i', j') \wedge \nwarrow a_I = (i, j)$.

The reason is similar to the reason for changing the condition in case (1).

4.5.5.2 Organizing Values

We now consider how to compute optimal alignment between $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$ without open stem pairs, i.e. $A(i_1, i_2; j_1, j_2|\emptyset)$.

We can create a table S of size $(i_2 - i_1 + 1)(j_2 - j_1 + 1)$. Each cell $S(i, j)$ ($i_1 - 1 \leq i \leq i_2$, $j_1 - 1 \leq j \leq j_2$) of the table contains $A(i_1, i; j_1, j|M)$. It is obvious that M may have multiple instances. So at each cell $S(i, j)$ of the table, we need to maintain multiple values, and each value corresponds to an instance of the item $A(i_1, i; j_1, j|M)$. We can use Lemma 4.5.9 to Lemma 4.5.11 to fill the table cell by cell, starting at the upper-left cell and scan the table from left to right, row by row as we are filling it. At each cell, we fill all possible values one by one. The alignment cost between $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$ would be in the lower-right cell.

An intuitive way to implement this approach is as follows. For all values at each position (i, j) ($i_1 - 1 \leq i \leq i_2$, $j_1 - 1 \leq j \leq j_2$), we store them in an array. Thus for the alignment between $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$, we need $(i_2 - i_1 + 1)(j_2 - j_1 + 1)$ arrays. However, this approach is not easy to implement, since it is hard to index multiple values of a position.

We can consider another way of organizing these values. Take a closer look at the item $A(i_1, i; j_1, j|M)$. Actually, we can organize values according to the set of proper open stem pairs M. For values of different cells which correspond to the same M, we put them into a matrix; then there is only one value at each cell of the matrix. Thus for each instance of M, we have a matrix holding those values of which the set of proper open stem pairs is M. We need to maintain multiple matrices. Actually, during the bottom up approach computation, when a new instance of M appears (new instance of M are created by case (5) of Lemma 4.5.11), we create a corresponding matrix. For each matrix T_k ($k \geq 1$), we save its proper open stem pairs set $T_k.M$, start position $(T_k.x_{start}, T_k.y_{start})$ and end position $(T_k.x_{end}, T_k.y_{end})$ along with it. The entry $T_k[i][j]$ of the matrix T_k whose proper open stem pairs set is M corresponds to the item $A(i_1, i; j_1, j|M)$.

Consider the plane whose dimensions correspond to the two RNA sequences. At the beginning of the computation, we create a matrix T_1 with start position $(T_1.x_{start}, T_1.y_{start}) = (i_1 - 1, j_1 - 1)$, end position $(T_1.x_{end}, T_1.y_{end}) = (i_2, j_2)$, and proper open stem pairs set $T_1.M = \emptyset$. We use Lemma 4.5.9 and Lemma 4.5.10 to initialize the first row and first column of T_1, i.e. compute boundary conditions. Then at each position (i, j) $(i_1 \leq i \leq i_2,\ j_1 \leq j \leq j_2)$, we compute all matrix entries at (i, j) by Lemma 4.5.11. The computation is from left to right and row by row.

Notice that during the computation, new matrices are produced by case (5) of Lemma 4.5.11. We now discuss how to generate a new matrix.

4.5.5.3 Generate New Matrices

Matrices are generated as needed and are indexed by the generated order.

We use $index_T$ to record the number of matrices that have been created. Assume that we are currently computing the entry (i, j) of the matrix T_k. If the modified condition of case (5) of Lemma 4.5.11 is satisfied, that is, if (i, j) corresponds to the left end of the innermost arc pair of a crossing stem pair (a_O, a_I) which is inside $(i_1, i_2; j_1, j_2)$, we check whether (a_O, a_I) is compatible with current proper open stem pairs set $T_k.M$. If (a_O, a_I) is compatible with $T_k.M$, we increase $index_T$ by 1 and create a new matrix T_{index_T} of which the proper open stem pairs set is $T_k.M \cup \{(a_O, a_I)\}$. We need to determine the size of this new matrix T_{index_T}.

We give definitions of the *open area of a crossing stem pair* and the *open area of a crossing stem pairs set M* as follows. These definitions are useful to determine the size of the matrices.

Consider a plane whose dimensions correspond to the two RNA sequences. We define the *open area of a crossing stem pair* (a_O, a_I) where $a_O = (a_{O_1}, a_{O_2})$ and $a_I = (a_{I_1}, a_{I_2})$[12] as the area enclosed by the rectangle with the upper-left corner $(a_{I_1}^L, a_{I_2}^L)$ and the lower-right corner $(a_{I_1}^R - 1, a_{I_2}^R - 1)$ (see Figure 4.5). The *open area*

[12] a_{O_1} and a_{I_1} are arcs from the first RNA, and a_{O_2} and a_{I_2} are arcs from the second RNA.

of a crossing stem pairs set M is the area shared by open areas of all stem pairs in M (see Figure 4.6).

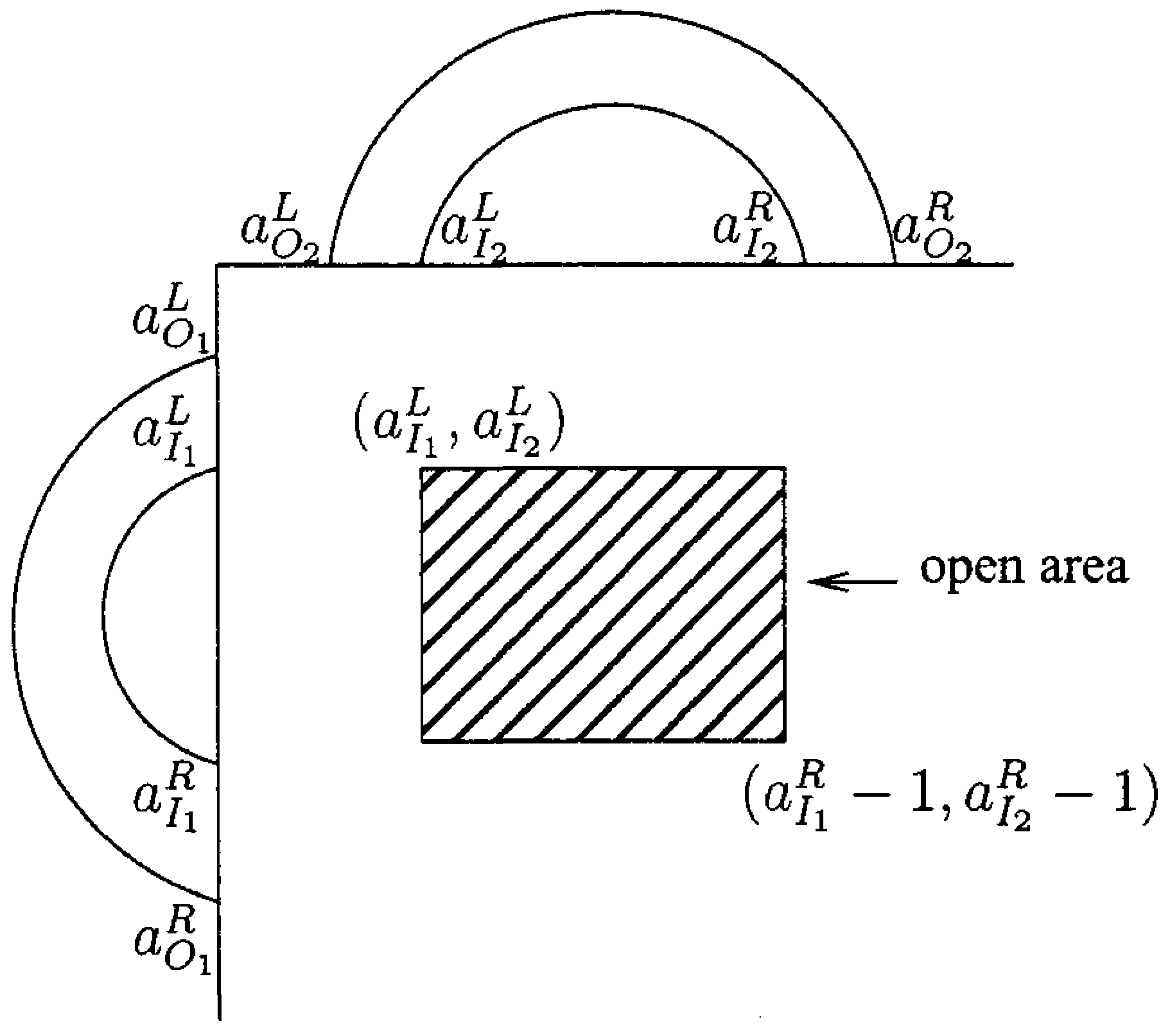


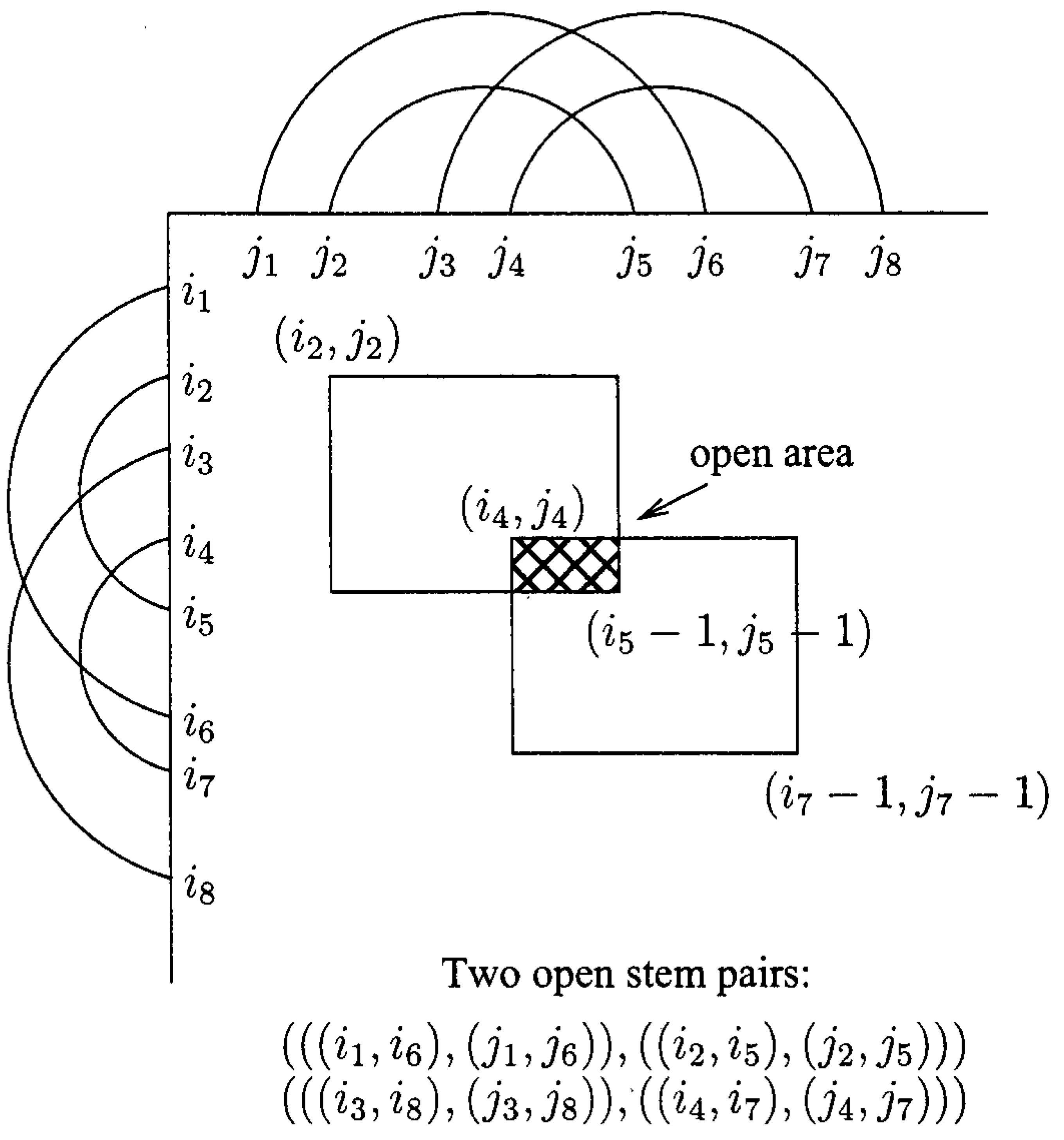
Figure 4.5: An illustration of open area of a crossing stem pair

We now consider how to determine the size of the new matrix T_{index_T}.

Obviously, the new matrix T_{index_T} corresponds to the open area of $T_k.M \cup \{(a_O, a_I)\}$. The start position of T_{index_T} is the upper-left corner of the open area of $T_k.M \cup \{(a_O, a_I)\}$ which is current position (i, j), and the end position is the lower-right corner of the open area of $T_k.M \cup \{(a_O, a_I)\}$. We need to determine the lower-right corner of the open area of $T_k.M \cup \{(a_O, a_I)\}$. It can be easily obtained in the following way.

We know that the open area of $T_k.M \cup \{(a_O, a_I)\}$ is the overlapped area of the open area of all stem pairs in $T_k.M \cup \{(a_O, a_I)\}$. Thus the open area of $T_k.M \cup \{(a_O, a_I)\}$ must be the overlapped area of the open area of $T_k.M$ and the open area of (a_O, a_I). The lower-right corner of the open area of $T_k.M$ is the end position $(T_k.x_{end}, T_k.y_{end})$ of T_k, and the lower-right corner of the open area of (a_O, a_I) is $(p_{r_1}(i) - 1, p_{r_2}(j) - 1)$[13]. Thus if $(T_k.x_{end}, T_k.y_{end}) \prec (p_{r_1}(i) - 1, p_{r_2}(j) - 1)$, $(T_k.x_{end}, T_k.y_{end})$ would be the

[13]Assume that $a_I = (a_{I_1}, a_{I_2})$, then $\searrow a_I = (a^R_{I_1}, a^R_{I_2}) = (p_{r_1}(i), p_{r_2}(j))$ since $\nwarrow a_I = (i, j)$. Thus the lower-right corner of the open area of (a_O, a_I) is $(p_{r_1}(i) - 1, p_{r_2}(j) - 1)$.



j_1 j_2 j_3 j_4 j_5 j_6 j_7 j_8
i_1 i_2 i_3 i_4 i_5 i_6 i_7 i_8
(i_2, j_2)
open area
(i_4, j_4)
$(i_5 - 1, j_5 - 1)$
$(i_7 - 1, j_7 - 1)$
Two open stem pairs:
$(((i_1, i_6), (j_1, j_6)), ((i_2, i_5), (j_2, j_5)))$
$(((i_3, i_8), (j_3, j_8)), ((i_4, i_7), (j_4, j_7)))$

Figure 4.6: An illustration of open area of a crossing stem pairs set

lower-right corner of the open area of $T_k.M \cup \{(a_O, a_I)\}$, that is the end position of T_{index_T}; otherwise, $(p_{r_1}(i) - 1, p_{r_2}(j) - 1)$ would be the lower-right corner of the open area of $T_k.M \cup \{(a_O, a_I)\}$, that is the end position of T_{index_T}.

After creating a new matrix T_{index_T}, we initialize the first row and first column of T_{index_T}. The upper-left entry $T_{index_T}[i][j]$ can be filled by case (5) of Lemma 4.5.11. The first row except the upper-left entry of T_k can be filled by case (1) of Lemma 4.5.11 and the first column except the upper-left entry can be filled by case (2) of Lemma 4.5.11 directly.

Notice that there may be more than one crossing stem pair of which the left end of the innermost arc pair is (i, j). We need to iterate over all possible alternatives, thus there may be multiple matrices created at (i, j).

We can always generate new matrices in this way.

4.5.5.4 Maintain *liveL* and *activeL*

At each position (i,j) $(i_1 \le i \le i_2,\ j_1 \le j \le j_2)$, we compute all matrix entries at (i,j) by Lemma 4.5.11. At (i,j), only some matrices exist. Thus we need to know which matrices exist at (i,j). We can maintain several lists of matrices indices to avoid unnecessary check. We say that a matrix is *active* at (i,j) if it exists at (i,j).

We use two lists *liveL* and *activeL*. Assume that the computation is currently at position (i,j). Then *liveL* stores the indices of matrices of which the end positions the computation has not arrived at. Formally, *liveL* contains all $k(k \ge 1)$ which satisfy that $i < T_k.x_{end}$, or $i = T_k.x_{end}$ and $j \le T_k.y_{end}$. *activeL* stores the indices of matrices which exist at (i,j) (if i is the first row or j is the first column of a matrix T_s, we do not add s to *activeL*, since the first row and first column of a matrix are initialized when the matrix is created). Formally, *activeL* contains all $k(k \ge 1)$ which satisfy that $(T_k.x_{start}, T_k.y_{start}) \prec (i,j) \preceq (T_k.x_{end}, T_k.y_{end})$.

When the computation moves into the matrices in *liveL* (except the first column and first row of the matrices), these matrices will become "active". Thus we can use *liveL* to generate *activeL*. We need a temporary list *tmpL* to generate *activeL* from *liveL*.

We can maintain these lists as follows.

When a matrix T_k is created, we append its index k to the *liveL* list. When the computation arrives the end position of T_k, after computing all values of current position, we search *liveL* to find T_k's index k, then delete k from *liveL*.

Before computing each row, we copy *liveL* to *tmpL*, and append 1 to *activeL* since we begin computing each row i from the position (i, j_1) where T_1 is already "active". When the first column of a matrix T_k is encountered, after computing all values of current position, we search *tmpL* to find T_k's index, then append it to *activeL* and delete it from *tmpL*. By doing this, we can skip the first column of matrix T_k which is already initialized (except T_1 whose first row and first column are already skipped). Notice that if $T_k.y_{start} = T_k.y_{end}$, that is, the matrix T_k is just a single column, we do not append k to *activeL*. When a matrix T_k is created, we do not append its index k to *activeL* either since the first column of matrix T_k is already initialized. When the last column of a matrix T_k is encountered, after computing all values of current position, we search *activeL* to find T_k's index, then delete it from *activeL*.

These lists can be implemented in linked list for fast insertion and deletion.

4.5.5.5 Computation of Lemma 4.5.11

At each position (i, j), for each matrix whose index is in *activeL*, we compute their values at (i, j) by Lemma 4.5.11. The first three cases of Lemma 4.5.11 are trivial. And we have already discussed case (5) which is used for generating new matrices in Section 4.5.5.3. Now we discuss how to compute cases (4) and (6).

Assume that we are currently computing the entry (i, j) of the matrix T_k. We first consider how to compute case (4) of Lemma 4.5.11.

If (i, j) is the right end of a non-crossing arc pair in NC inside $(i_1, i_2; j_1, j_2)$, we need to consider aligning this arc pair using case (4) of Lemma 4.5.11 if this arc pair is compatible with current proper open stem pairs set $T_k.M$. Assume that this arc pair is $a \in NC$ with $\searrow a = (i, j)$, and it is compatible with current proper open stem pairs set $T_k.M$.

First, we need to determine M_1 and M_2 in case (4) of Lemma 4.5.11. We can partition current proper open stem pairs set $T_k.M$ into two subsets: M' contains each stem pair whose left end is before the position $(p_{r_1}(i), p_{r_2}(j))$; M'' contains each stem pair whose left end is after the position $(p_{r_1}(i), p_{r_2}(j))$. It is easy to see that $M_1 = M' \cup M_{share}$ and $M_2 = M'' \cup M_{share}$. M' and M'' are uniquely determined. Thus we only need to compute M_{share}. Obviously, M_{share} may have multiple instances. We need to enumerate all possible alternatives of M_{share}. Actually, all instances of M_{share} are in the list of all possible proper open stem pairs sets of the segment $(p_{r_1}(i) + 1, i - 1; p_{r_2}(j) + 1, j - 1)$ which we have precomputed in Algorithm 4.3. Thus we can search M_{share} in this list. After we get M_{share}, we can easily obtain M_1 and

M_2.

After we obtain M_1 and M_2, we need to find the score $A(i_1, p_{r_1}(i) - 1; j_1, p_{r_2}(j) - 1|M_1)$ and the subalignment score $A(p_{r_1}(i) + 1, i - 1; p_{r_2}(j) + 1, j - 1|M_2)$. For $A(i_1, p_{r_1}(i) - 1; j_1, p_{r_2}(j) - 1|M_1)$, we need to find the matrix which contains the score $A(i_1, p_{r_1}(i) - 1; j_1, p_{r_2}(j) - 1|M_1)$. Therefore, we can search the index of the matrix whose proper open stem pairs set is M_1. For $A(p_{r_1}(i)+1, i-1; p_{r_2}(j)+1, j-1|M_2)$, we need to search it in the alignment scores of the segment $(p_{r_1}(i)+1, i-1; p_{r_2}(j)+1, j-1)$ which are computed and saved before computing the subalignment between $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$. Therefore, we can search the index of the precomputed subalignment score whose corresponding subalignment's proper open stem pairs set is M_2 in the saved alignment score array of the segment $(p_{r_1}(i) + 1, i - 1; p_{r_2}(j) + 1, j - 1)$.

Notice that for non-crossing arc pairs covered by the same non-crossing *m_stem* pair, the instances of M_1 and M_2 do not change. Thus we can only compute all possible corresponding indices for M_1 and M_2 when the innermost arc pair of a non-crossing *m_stem* pair is encountered, and save them for later use. When other arc pairs which are not innermost arc pair of a non-crossing *m_stem* pair is encountered, we can get those indices that we need directly.

We now consider how to compute case (6) of Lemma 4.5.11.

If (i, j) is the right end of the outermost arc pair of a crossing stem pair in ST_{CR} inside $(i_1, i_2; j_1, j_2)$, we need to consider computing case (6) of Lemma 4.5.11 if this stem pair is compatible with current proper open stem pairs set $T_k.M$. Assume that this stem pair is $(a_O, a_I) \in ST_{CR}$ with $\searrow a_O = (i, j)$, and it is compatible with current proper open stem pairs set $T_k.M$.

We need to find the score $A(i_1, i' - 1; j_1, j' - 1|M \cup \{(a_O, a_I)\}$ in case (6). Thus we need to find the matrix which contains the score $A(i_1, i' - 1; j_1, j' - 1|M \cup \{(a_O, a_I)\}$. Therefore, we can search the index of the matrix whose proper open stem pairs set is $M \cup \{(a_O, a_I)\}$.

In fact, case (6) is used to merge ending score of the matrix with proper open stem pairs set $M \cup \{(a_O, a_I)\}$ to its corresponding matrix with proper open stem pairs set

M.

Notice that there may be more than one crossing stem pair in ST_{CR} of which the right end of the outermost arc pair is (i, j). We need to iterate over all possible alternatives, and take the minimum of all these alternatives.

4.5.6 Algorithm for Computing Optimal Subalignment with Open Stem pairs

In this section, we discuss how to compute optimal alignment between $R_1[i_1, i_2]$ and $R_2[j_1, j_2]$ with open stem pairs, i.e. $A(i_1, i_2; j_1, j_2|M_0)$ ($M_0 \neq \emptyset$). The computation of $A(i_1, i_2; j_1, j_2|M_0)$ ($M_0 \neq \emptyset$) is more complicated than the computation of $A(i_1, i_2; j_1, j_2|\emptyset)$, but is similar. It can be considered as a constrained alignment problem where our goal is to find the optimal subalingment using stem pairs in M_0 as the constraints.

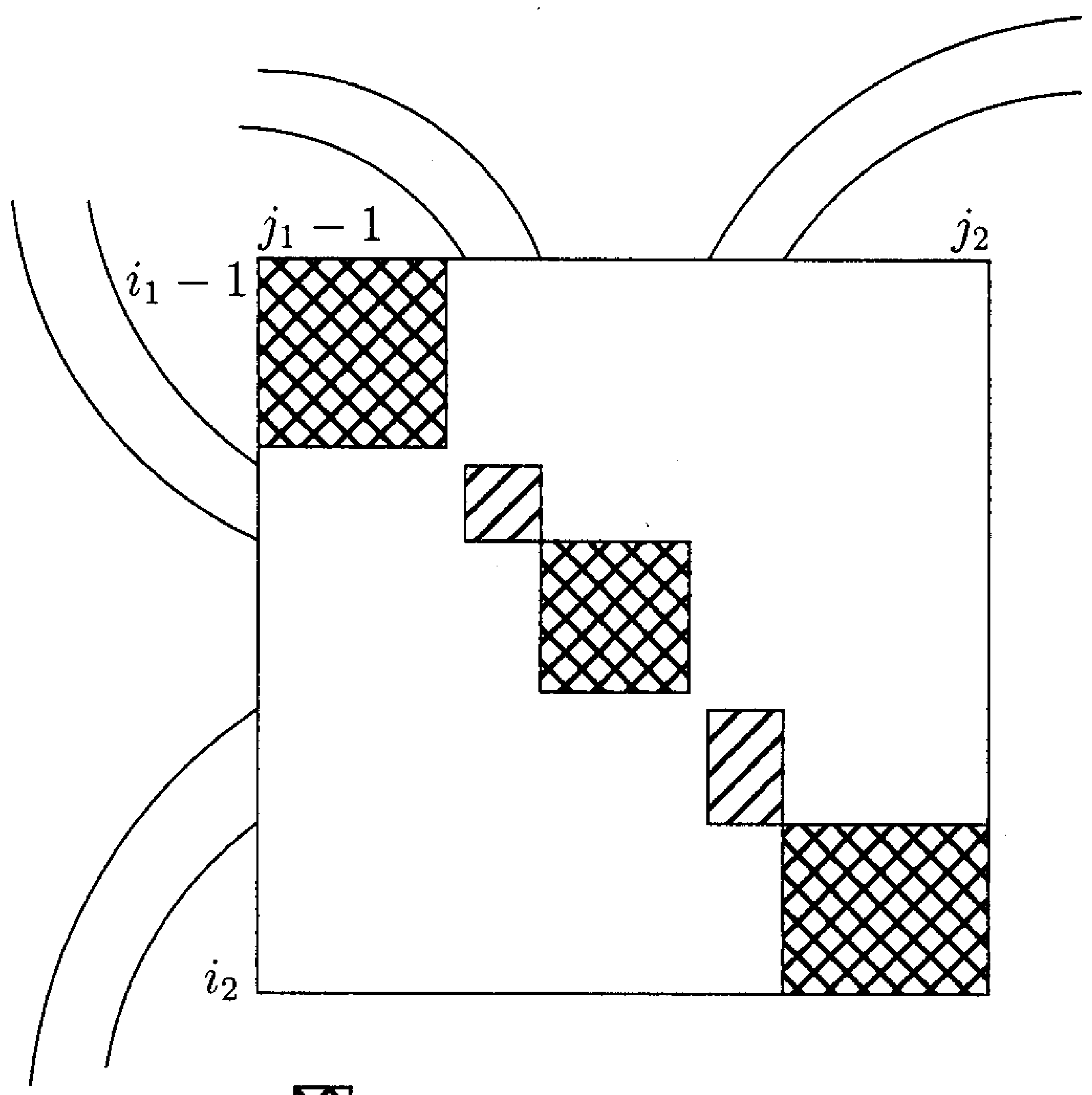
The (left or right) ends of the outermost arc pairs or innermost arc pairs of stem pairs in M_0 will partition the plane whose dimensions correspond to the two RNA sequences into several parts. Figure 4.7 gives a simple illustration. We only need to compute the shaded region in this figure.

Now we discuss how to compute $A(i_1, i_2; j_1, j_2|M_0)$ ($M_0 \neq \emptyset$).

We first compute two sorted lists of *partition points* formed by the ends (left or right) of the outermost arc pairs or innermost arc pairs of stem pairs in M_0. These lists are $parPoints_1$ and $parPoints_2$. Points in these two lists are sorted by the partial order "$\prec$" which is defined in Section 4.1.

The first list $parPoints_1$ consists of the right ends of the innermost arc pairs of the stem pairs in M_0 whose right ends are inside $(i_1, i_2; j_1, j_2)$ and the left ends of the outermost arc pairs of the stem pairs in M_0 whose left ends are inside $(i_1, i_2; j_1, j_2)$.

The second list $parPoints_2$ consists of the right ends of the outermost arc pairs of the stem pairs in M_0 whose right ends are inside $(i_1, i_2; j_1, j_2)$ and the left ends of the innermost arc pairs of the stem pairs in M_0 whose left ends are inside $(i_1, i_2; j_1, j_2)$.



▩ the region we need to compute

▨ precomputed open stem pair alignment

Figure 4.7: An illustration of computing $A(i_1, i_2; j_1, j_2|M_0)$ $(M_0 \neq \emptyset)$

We can also add the point (i_1-1, j_1-1) to $parPoints_2$ and (i_2, j_2) to $parPoints_1$ to make the algorithm easy to be implemented.

Assume that $|parPoints_1| = |parPoints_2| = num_{par}$. Then for each $k \in \{1, \cdots, num_{par}\}$, we need to compute the area enclosed by the rectangle with the upper-left corner $parPoints_2[k]$ and the lower-right corner $parPoints_1[k]$. The alignment scores at $parPoints_2[k]$ $(1 \le k \le num_{par})$ can be computed by case (5) of Lemma 4.5.11. Notice that we do not create new matrices for the stem pairs in M_0 since they are required to be realized.

The computation of $A(i_1, i_2; j_1, j_2|M_0)$ $(M_0 \neq \emptyset)$ is similar to the computation of $A(i_1, i_2; j_1, j_2|\emptyset)$. We can extend the algorithm for computing $A(i_1, i_2; j_1, j_2|\emptyset)$ here. In the following text, we will point out the differences.

For the computation of case (4) of Lemma 4.5.11 in computing $A(i_1, i_2; j_1, j_2|M_0)$, we need to check whether the arc pair $((p_{r_1}(i), i), (p_{r_2}(j), j))$ is compatible with M_0 first. When generating M_{share}, we also need to check whether M_{share} is compatible with M_0. Other computation is the same as the computation of case (4) of Lemma 4.5.11 in computing $A(i_1, i_2; j_1, j_2|\emptyset)$.

For computation of cases (5) and (6) of Lemma 4.5.11 in computing $A(i_1, i_2; j_1, j_2|M_0)$, we need to check whether the stem pair (a_O, a_I) is compatible with M_0 first. Other computation is the same as the computation of cases (5) and (6) of Lemma 4.5.11 in computing $A(i_1, i_2; j_1, j_2|\emptyset)$.

The method to maintain lists *activeL* and *tmpL* also changes a little. Before computing each row, for each index k' in *liveL*, we do the following step: if the first column of the matrix $T_{k'}$ is not before the first column that we need to compute (see Figure 4.7; the first column of the shaded region), then we append k' to *tmpL*; if the first column of the matrix $T_{k'}$ is before the first column that we need to compute and the last column of $T_{k'}$ is not before the first column that we need to compute, then we append k' to *activeL*. After computing each row, we clear lists *activeL* and *tmpL*. Other operations to maintain those lists remain the same.

Actually, we can use this algorithm to compute $A(i_1, i_2; j_1, j_2|\emptyset)$. It is a generalization of the algorithm for computing $A(i_1, i_2; j_1, j_2|\emptyset)$.

4.5.7 Algorithm for Computing Optimal Global Alignment

The cost of the global alignment is the value of $Align(R_1, R_2) = A(1, |R_1|; 1, |R_2||\emptyset)$. We can compute $A(1, |R_1|; 1, |R_2||\emptyset)$ using a bottom-up approach.

Given two RNA structures R_1 and R_2, we can first partition the set of *m_stem* pairs $ST_1^{MAX} \times ST_2^{MAX}$ into the set of non-crossing *m_stem* pairs ST_{NC}^{MAX} and the set of crossing *m_stem* pairs ST_{CR}^{MAX}. Then we generate NC from ST_{NC}^{MAX}.

Then we preprocess crossing stem pairs by methods discussed in Section 4.5.2 and get the filtered crossing stem pairs set ST_{CR}.

Then we use Algorithm 4.3 to generate lists of all possible proper open stem pairs sets for segments enclosed by the innermost arc pairs of non-crossing *m_stem* pairs in ST_{NC}^{MAX}.

From case (4) of Lemma 4.5.11, we only need to compute alignment of segment $(i_1, i_2; j_1, j_2)$ such that $(i_1 - 1, i_2 + 1; j_1 - 1, j_2 + 1)$ is a non-crossing arc pair in NC.

For each arc pair $((i_1, i_2), (j_1, j_2)) \in NC$, we can obtain a list of all possible proper open stem pairs sets M_{list} of its corresponding segment $(i_1 + 1, i_2 - 1; j_1 + 1, j_2 - 1)$ from the output of Algorithm 4.3. Then we compute all possible $A(i_1 + 1, i_2 - 1; j_1 + 1, j_2 - 1|M_0)$ $(M_0 \in M_{list})$.

Finally, we compute $A(1, |R_1|; 1, |R_2||\emptyset)$.

4.5.8 Trace Back to Produce Optimal Alignment

In biological applications, in addition to the optimal alignment score between two RNA structures, it is often required to produce an alignment corresponding to the optimal score. The method employed to produce an alignment corresponding to the optimal score is called *traceback*. We have introduced the traceback method for sequence alignment in Chapter 2. We start from the cell position holding the optimal score in the alignment score matrices which is also the end position in the alignment, then tracing back to the start position of the alignment to produce the whole alignment. Generally, there are two approaches for traceback: (1) saving pointers while computing alignment scores; (2) recomputation. In this thesis, we use the second approach to save space. Starting from the matrix cell holding the optimal score, we repeat the recurrence formulae and check to see which direction results in the given score. When there are more than one directions could result the given score, we select one direction to trace back. In this thesis, we set a preference order, from the highest to the lowest, is: insertion (case (1) of Lemma 4.5.11) $>$ deletion (case (2) of Lemma 4.5.11) $>$ base substitution (case (3) of Lemma 4.5.11) $>$ base pair substitution (cases (4), (5) and (6) of Lemma 4.5.11).

We can do the traceback by a top-down approach. Starting from the position $(|R_1|, |R_2|)$, we first do a top-layer traceback without going down further into the sublayers. We use four stacks to keep the information that we need to produce optimal global alignment. Assume that the optimal alignment is $Align$. $stack_1$ contains sequence indices in the resulting alignment and a sign number (1, 2, 3 or 4) which indicates the next traceback operation. $stack_2$ contains proper open stem pairs lists of subalignments which need to be unraveled next. $stack_3$ contains the indices of crossing stem pairs in ST_{CR} whose left parts need to be unraveled next. $stack_4$ contains the indices of crossing stem pairs in ST_{CR} whose right parts need to be unraveled next.

When we encounter a situation where $R_1[i]$ is deleted, we push $(i, -1, 1)$ to $stack_1$. When we encounter a situation where $R_2[j]$ is inserted, we push $(-1, j, 1)$ to $stack_1$. When we encounter a situation where $R_1[i]$ is aligned to $R_2[j]$ but no arc pair involving $(R_1[i], R_2[j])$ is realized, we push $(i, j, 1)$ to $stack_1$. The first two numbers in each entry of $stack_1$ denote sequence indices in an alignment. The third number "1" indicates that these indices can be converted to corresponding characters and appended to the alignment.

When we encounter a situation where (i, j) is the right end of a non-crossing arc pair and this arc pair is realized, we push $(i, j, 2)$ to $stack_1$, and push the proper open stem pairs list of the subalignment enclosed by this arc pair to $stack_2$. The third number "2" in the entry of $stack_1$ indicates that there is a subalignment $Align[p_{r_1}(i) + 1, i - 1; p_{r_2}(j) + 1, j - 1]$ which need to be unraveled next. $stack_2$ contains its corresponding proper open stem pairs list.

When we encounter a situation where (i, j) is the right end of the outermost arc pair of a crossing stem pair and this stem pair is realized, we push $(i, j, 4)$ to $stack_1$, and push the index of this stem pair in the sorted (by $5'$ ends) list ST_{CR} of crossing stem pairs to $stack_4$. The third number "4" in the entry of $stack_1$ indicates that there is a stem pair alignment whose right part need to be unraveled next.

When we encounter a situation where (i, j) is the left end of the innermost arc pair of a crossing stem pair and this stem pair is realized, we push $(i, j, 3)$ to $stack_1$, and push the index of this stem pair in the sorted (by $5'$ ends) list ST_{CR} of crossing stem pairs to $stack_3$. The third number "3" in the entry of $stack_1$ indicates that there is a stem pair alignment whose left part need to be unraveled next.

After this top-layer traceback, we pop the stack entries from $stack_1$. Assume the first two numbers are i and j.

If the sign number, that is the third number of the entry is 1, we convert the first two numbers i and j to corresponding characters and appended to the alignment.

If the sign number of the entry is 2, then there is a pair of substructures $R_1[p_{r_1}(i) + 1, i - 1]$ and $R_2[p_{r_2}(j) + 1, j - 1]$ that need to be unraveled. We push $(i, j, 1)$ to $stack_1$. Then we pop stack entry from $stack_2$ to get the proper open stem pairs list for this segment. Then we recompute alignment between $R_1[p_{r_1}(i) + 1, i - 1]$ and $R_2[p_{r_2}(j) + 1, j - 1]$, and perform a next layer traceback for $R_1[p_{r_1}(i) + 1, i - 1]$ and $R_2[p_{r_2}(j) + 1, j - 1]$ as what we do in the top-layer traceback. When traceback returns, we have unraveled the alignment in the next layer. Then the next item popped from $stack_1$ would be the one we pushed back to $stack_1$ immediately before we performed the previous traceback. Since we changed its sign number to 1, it can be output directly now.

If the sign number of the entry is 3, then there is a crossing stem pair alignment whose left part need to be unraveled. We pop stack entry from $stack_3$ to get the index of this stem pair. Then we recompute alignment of this stem pair, and perform a next layer traceback for this stem pair. When traceback returns, we will get the alignment of this stem pair. We append the left part of this stem pair alignment to the global alignment, and save the right part of this stem pair alignment for later use.

If the sign number of the entry is 4, then there is a crossing stem pair alignment whose right part need to be unraveled. We pop stack entry from $stack_4$ to get the index of this stem pair. Then we can append the right part of this stem pair alignment which we have saved previously to the global alignment.

This procedure repeats until $stack_1$ is empty. Then we have the resulting alignment.

4.5.9 Complexity

Let n be $\max(|R_1|, |R_2|)$, and let s and t be the maximal number of arcs and bases in a stem, respectively. Recall that we use ST_1^{MAX} and ST_2^{MAX} to denote the *m_stems* sets of R_1 and R_2, respectively; we use P_1 and P_2 to denote the arcs sets of R_1 and R_2, respectively.

The first step of the algorithm is the partition of *m_stem* pairs using methods discussed in Section 4.5.1. For each partition method, we need at most $O(n^2)$ time, and at most $O(n^4)$ time to optimize the preliminary partition result. The space we need is at most $O(n^2)$. We also generate NC from ST_{NC}^{MAX} in this step. This takes at most $O(n^2)$ time and space. Therefore, in the first step, we need at most $O(n^4)$ time and $O(n^2)$ space.

Next we preprocess crossing stem pairs using Algorithm 4.2. For each crossing inner local *m_stem* pair (a_O, a_I), there are $O(n^2)$ possible instances of a_I, and a_O is uniquely determined by a_I. Thus we have $O(n^2)$ crossing inner local *m_stem* pairs to compute. For each crossing inner local *m_stem* pair, we compute its alignment by Algorithm 4.1. Computation of each crossing inner local *m_stem* pair will need $O(t^4)$ time and space. Recall we also record realized stem pairs. Obviously, in the computation of each crossing inner local *m_stem* pair, there are at most s stem pairs which are realized at the same time. Notice that we compute each crossing inner local *m_stem* pair one by one such that we only need at most $O(t^4 + n^2s)$ space. Thus we need at most $O(n^2t^4)$ time and $O(t^4 + n^2s)$ space in this step. We can also compute alignment of all crossing outer local *m_stem* pairs. This also takes at most $O(n^2t^4)$ time and $O(t^4 + n^2s)$ space. These two approaches will produce two sorted filtered crossing stem pairs lists, each of which has at most $O(n^2s)$ stem pairs. We compute the intersection of these two lists. This takes at most $O(n^2s \cdot log(n^2s))$ time. Therefore, in the step of preprocessing crossing stem pairs, we need at most $O(n^2t^4 + n^2s \cdot log(n^2s))$ time and $O(t^4 + n^2s)$ space.

Then we use Algorithm 4.3 to generate lists of all possible proper open stem pairs sets for segments enclosed by the innermost arc pairs of non-crossing *m_stem* pairs in ST_{NC}^{MAX}. We adopt the notion of the *crossing number of a position (x, y)* proposed by Möhl *et al.* which is introduced on page 55 in Section 3.4.4 to measure the number of proper open stem pairs. The maximal crossing number is denoted as k. Consider a non-crossing *m_stem* pair $((p_{O_1}, p_{O_2}), (p_{I_1}, p_{I2}))$. Let $i_1 = p_{I_1}^L + 1$, $i_2 = p_{I_1}^R - 1$, $j_1 = p_{I_2}^L + 1$ and $j_2 = p_{I_2}^R - 1$. For each crossing *m_stem* pair b which is open for the segment $(i_1, i_2; j_1, j_2)$, there are $O(s \cdot s + \binom{s}{2} \cdot \binom{s}{2}) = O(s^4)$ crossing stem pairs that are covered by b. These crossing stem pairs can be realized and serve as proper open stem pairs for the alignment of segment $(i_1, i_2; j_1, j_2)$. Therefore, there are at most $O((s^4)^{C(i_1,j_1)} \cdot (s^4)^{C(i_2,j_2)}) = O(s^{8k})$ possible proper open stem pairs for the segment $(i_1, i_2; j_1, j_2)$. Therefore, for each non-crossing *m_stem* pair c, we will use $O(s^{8k})$ time and space to compute and save all possible proper open stem pairs lists of segments enclosed by non-crossing arc pairs that are covered by c. Totally, we will use $O(|ST_{NC}^{MAX}|s^{8k}) = O(n^2s^{8k})$ time and space in this step.

For each non-crossing arc pair $((i_1, i_2), (j_1, j_2)) \in NC$, we compute all possible $A(i_1 + 1, i_2 - 1; j_1 + 1, j_2 - 1|M_0)$ $(M_0 \in M_{list})$ where M_{list} is the list of all possible proper open stem pairs sets of the segment $(i_1 + 1, i_2 - 1; j_1 + 1, j_2 - 1)$. M_{list} is precomputed in last step and it has at most $O(s^{8k})$ elements. Thus for each segment enclosed by a non-crossing arc pair, we need to compute $O(s^{8k})$ subalignments. Since there are $O(n^2)$ non-crossing arc pairs, we need to compute $O(n^2s^{8k})$ subalignments.

Now we consider the time and space complexities to compute a subalignment $A(i_1, i_2; j_1, j_2|M_0)$ where M_0 is the proper open stem pairs set of the segment $(i_1, i_2; j_1, j_2)$ and is fixed here. We need to compute all $A(i_1, i; j_1, j|M)$ $(i_1 - 1 \leq i \leq i_2, j_1 - 1 \leq j \leq j_2)$ where M is the proper open stem pairs set at position (i, j). We need to measure the number of instances of M. Since M_0 is fixed, we only need to consider stem pairs that are inside $(i_1, i_2; j_1, j_2)$ and cross the position (i, j). Thus there are at most $O((s^4)^{C(i,j)}) = O(s^{4k})$ instances of M. That means there are at most $O(s^{4k})$ values that we need to compute at each position. At each position (i, j), for each copy of $A(i_1, i; j_1, j|M)$, we compute its value by Lemma 4.5.11. The first three cases needs constant time. For case (4), we need to compute all possible instances of M_1 and M_2. Since M_1 and M_2 are uniquely determined by M_{share}, we only need to consider the number of instances of M_{share}. Obviously, there are at most $O((s^4)^{C(i,j)}) = O(s^{4k})$ instances of M_{share}. Thus there are at most $O(s^{4k})$ instances of M_1 and M_2. Hence in case (4), we need $O(s^{4k})$ time. When the left end of the innermost arc pair of a crossing stem pair is encountered, new instances of M are created. We need to iterate over all possible $O(s^2)$ alternatives of the outermost arc pair of this stem pair. For each new instance M, we need constant time according to case (5) of Lemma 4.5.11. Hence we need at most $O(s^2)$ time in this situation. Similarly, when the right end of the outermost arc pair of a crossing stem pair is encountered, we need to compute case (6). This also requires at most $O(s^2)$ time. Therefore, we need at most $O(n^2 \cdot s^{4k} \cdot s^{4k}) = O(n^2s^{8k})$ time for $O(n^2 \cdot s^{4k})$ values of $O(n^2)$ positions. We need $O(n^2s^{4k})$ space to hold these values. Notice that we save all possible proper open stem pairs lists of segments enclosed by non-crossing arc pairs computed as by Algorithm 4.3 and we will use the saved result in this step. This takes $O(n^2s^{8k})$ space. Thus we need at most $O(n^2s^{8k})$ time and $O(n^2s^{8k})$ space in total.

Finally, we compute $A(1, |R_1|; 1, |R_2||\emptyset)$. This also requires at most $O(n^2s^{8k})$ time and $O(n^2s^{8k})$ space.

So the time complexity of the whole algorithm for computing optimal alignment score between two RNA tertiary structures is $O(n^2s^{8k} \cdot n^2s^{8k}) = O(n^4s^{16k})$, and the space complexity is $O(n^2s^{8k})$.

Now we consider the traceback part.

We trace back from position $(|R_1|, |R_2|)$ to $(0, 0)$. At each position, we repeat the recurrence formulae and check to see which direction results in the given score. There are only three directions to go: upper, left, or upper-left. Thus we only need to go through at most $O(2n) = O(n)$ positions. At each position that is not inside the region enclosed by left ends or right ends of a realized crossing stem pair, we need at most $O(s^{4k} \cdot s^{4k}) = O(s^{8k})$ time (there are at most $O(s^{4k})$ values at each position, and each value needs at most $O(s^{4k})$ time). At each position that is inside the region enclosed by left ends or right ends of a realized crossing stem pair, we only need constant time. Thus we need at most $O(ns^{8k})$ time for traceback. Since we save all possible proper open stem pairs lists of segments enclosed by non-crossing arc pairs computed as by Algorithm 4.3 and this takes $O(n^2s^{8k})$ space, we still need $O(n^2s^{8k})$ space.

Therefore, the time complexity of the whole algorithm for computing optimal alignment between two RNA tertiary structures is $O(n^4s^{16k})$, and the space complexity is $O(n^2s^{8k})$.

Even though the worst case time and space complexities of our algorithm are the same as Möhl *et al.*'s, we will show that because of our improvement, our algorithm could use much less resources (time and space) in practice to compute optimal alignment between two RNA tertiary structures in next chapter.

4.5.10 Possible Further Optimization

For the segment $(i_1, i_2; j_1, j_2)$ that satisfies $((i_1 - 1, i_2 + 1), (j_1 - 1, j_2 + 1)) \in NC$, we compute all possible instances of $A(i_1, i_2; j_1, j_2|M_0)$.

For each instance of $A(i_1, i_2; j_1, j_2|M_0)$ ($M_0 \neq \emptyset$), we denote the partition point we encounter first as (i_0, j_0). We found that there were redundant computations for positions from (i_1-1, j_1-1) to (i_0, j_0) compared to the computation of $A(i_1, i_2; j_1, j_2|\emptyset)$ (see Figure 4.8).

We can do as follows to avoid redundant computation.

During the computation of $A(i_1, i_2; j_1, j_2|\emptyset)$, when we encounter the first partition point of each instance of $A(i_1, i_2; j_1, j_2|M_0)$, we record all scores of current position. When we compute $A(i_1, i_2; j_1, j_2|M_0)$, we start computation from its first partition point, and get initial scores from the corresponding scores we record previously.

Unfortunately, this optimization technique is not easy to implement. Our current implementation does not include it.

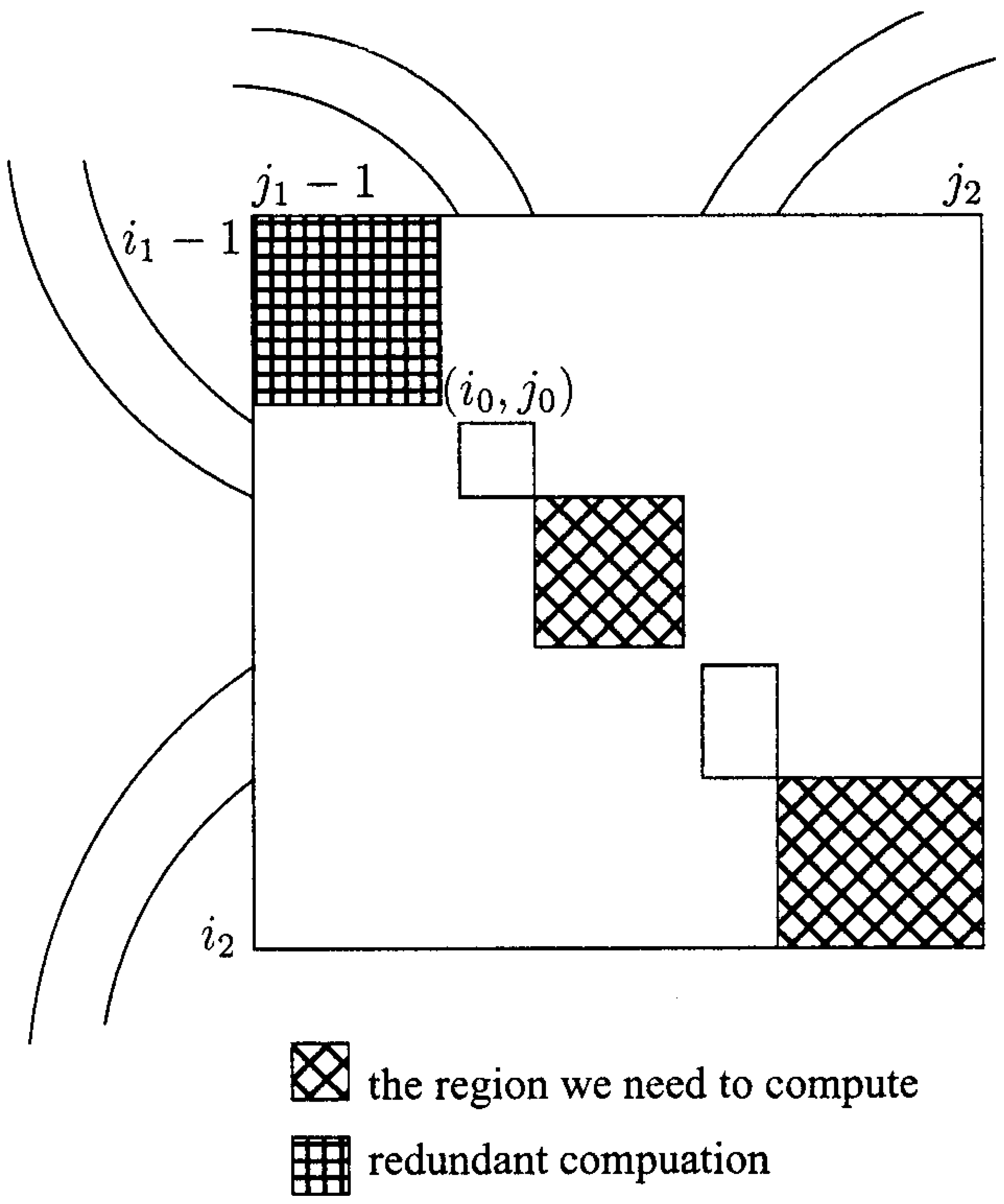


Figure 4.8: An illustration of redundant computation

4.6 Constrained Alignment

For simple RNA tertiary structures, we can compute the optimal alignment efficiently by algorithms discussed in Section 4.5. For moderate and complicated RNA tertiary structures, we need to consider other approaches to compute the alignment since using algorithms in Section 4.5 will cause high usage of space.

Like Wang and Zhang's RNA alignment algorithm that we have discussed in Section 3.3, we can adopt the constrained alignment approach. The method is as follows.

We select two crossing *m_stems* q_1 and q_2 which are very likely to be matched from two RNAs, respectively. These two stems form a stem pair. We impose the constraint, q_1 can only match to q_2 and q_2 can only match to q_1, on the alignment. We call the stem pair formed by q_1 and q_2 *constrained stem pair.*

In the first step, we align the constrained stem pair. Then we compute the alignment of the two RNAs.

We can align the constrained stem pair by Lemma 4.5.7 and Lemma 4.5.8 with a little modification. For the constrained stem pair, we do not require its innermost arc pair and outermost arc pair to be realized. Assume the constrained stem pair is (a_O, a_I) with $a_O = ((i_O, p_{r_1}(i_O)), (j_O, p_{r_2}(j_O)))$ and $a_I = ((i_I, p_{r_1}(i_I)), (j_I, p_{r_2}(j_I)))$. Then we start computation from the position $(i_I + 1, p_{r_1}(i_I) - 1; j_I + 1, p_{r_2}(j_I) - 1)$. Since we only have one constrained stem pair, we do not need a_I in S' item. Thus in Lemma 4.5.7, we need to change Eq. 4.21 to the following equation.

$$S'(i_I + 1, p_{r_1}(i_I) - 1; j_I + 1, p_{r_2}(j_I) - 1) = 0 \tag{4.27}$$

In Lemma 4.5.8, we only need to change the condition to "For $i_O \le i \le i_I + 1$, $j_O \le j \le j_I + 1$, $p_{r_1}(i_I) - 1 \le i' \le p_{r_1}(i_O)$, $p_{r_2}(j_I) - 1 \le j' \le p_{r_2}(j_O)$, and $(i, j, i', j') \ne (i_I + 1, j_I + 1, p_{r_1}(i_I) - 1, p_{r_2}(j_I) - 1)$", and discard a_I in the S' items in Eq. 4.22. Using the modified equations, we can align the constrained stem pair easily within

$O(t^4)$ time and space where t is the maximal number of bases in a stem.

Then we compute the alignment of the two RNAs using algorithms for computing optimal alignment discussed in the previous sections. Notice that after partitioning the set $ST_1^{MAX} \times ST_2^{MAX}$ of *m_stem* pairs, we need to filter out stem pairs in $ST_1^{MAX} \times ST_2^{MAX}$, ST_{CR}^{MAX} and ST_{NC}^{MAX} which are not compatible with the constrained stem pair. Then we use the filtered stem pairs sets in later steps of the algorithm.

The most part of the algorithm for computing constrained alignment is the same as the algorithm for computing optimal alignment. There is a difference that we need to discuss here. When we compute a subalignment $A(i_1, i_2; j_1, j_2|M_0)$ where M_0 is the proper open stem pairs set of the segment $(i_1, i_2; j_1, j_2)$ and is fixed. We need to add the ends which are inside $(i_1, i_2; j_1, j_2)$ of the constrained stem pair (a_O, a_I) into the partition points lists of the segment $(i_1, i_2; j_1, j_2)$. Thus we need to determine the relation between the constrained stem pair (a_O, a_I) and a segment $(i_1, i_2; j_1, j_2)$.

We can divide the relation between the constrained stem pair (a_O, a_I) and a segment $(i_1, i_2; j_1, j_2)$ into four types as follows.

(1) (a_O, a_I) is before $(i_1, i_2; j_1, j_2)$, or (a_O, a_I) is after $(i_1, i_2; j_1, j_2)$, or the left part of (a_O, a_I) is before $(i_1, i_2; j_1, j_2)$ and the right part of (a_O, a_I) is after $(i_1, i_2; j_1, j_2)$. Then the computation of $A(i_1, i_2; j_1, j_2|M_0)$ is exactly the same as what we do in computing optimal alignment.

(2) Both the left and right parts of (a_O, a_I) are inside $(i_1, i_2; j_1, j_2)$. Then we need to add the left end of a_O and the right end of a_I to the partition points list $parPoints_1$, and add the left end of a_I and the right end of a_O to the partition points list $parPoints_2$. Then we compute $A(i_1, i_2; j_1, j_2|M_0)$. This can be easily done by modifying the algorithm for computing optimal subalignment with open stem pairs a little.

(3) The left part of (a_O, a_I) is inside $(i_1, i_2; j_1, j_2)$ and the right part of (a_O, a_I) is after $(i_1, i_2; j_1, j_2)$. Then we need to add the left end of a_O to the partition points list $parPoints_1$, and add the left end of a_I to the partition points list $parPoints_2$. Then we compute $A(i_1, i_2; j_1, j_2|M_0)$.

(4) The left part of (a_O, a_I) is before $(i_1, i_2; j_1, j_2)$ and the right part of (a_O, a_I) is inside $(i_1, i_2; j_1, j_2)$. Then we need to add the right end of a_I to the partition points list $parPoints_1$, and add the right end of a_O to the partition points list $parPoints_2$. Then we compute $A(i_1, i_2; j_1, j_2|M_0)$.

For the traceback part, we can also modify the traceback algorithm for producing optimal alignment a little.

Although the result produced by constrained alignment is not guaranteed to be an optimal solution, in practice it would be reasonable.

Chapter 5

Implementation and Experiment Results

5.1 Implementation

A software package has been written to compute global alignment between two RNA structures. This package implements algorithms discussed in Chapter 4 and is written in ANSI C.

Two alignment options are provided: optimal alignment and constrained alignment discussed in Section 4.6. User can select one of them. For the constrained alignment option, the program will ask user to select two crossing maximal stems which seem very likely to be matched from two input RNAs, respectively, and input the start (first $5'$) base's positions of the two selected stems.

Three input files are needed.

The first input file specifies score scheme for single bases. Table 5.1 show an example of this file. The first column is the alphabet used to align unpaired base in R_1, while the first row is the alphabet used to align bases in R_2. We use D to represent space symbol $'-'$ in the input file. We treat the first column and the first row as indices of the table formed by numeric values in the file. The value of entry

(a, b) of the table $(a, b \in \{A, C, G, U, D\})$ is the cost of substituting a with b if $a \neq D$ and $b \neq D$; or the cost of deleting a if $a \neq D$ and $b = D$; or the cost of inserting b if $a = D$ and $b \neq D$. (Notice that the entry (D, D) has no meaning since we cannot align '$-$' to '$-$'.)

	A	C	G	U	D
A	0	1	1	1	1
C	1	0	1	1	1
G	1	1	0	1	1
U	1	1	1	0	1
D	1	1	1	1	0

Table 5.1: An example of score scheme for single bases

The second input file specifies score scheme for base pairs. Table 5.2 show an example of this file. The first column is the alphabet used to align base pairs in R_1, while the first row is the alphabet used to align base pairs in R_2. We use DD to represent $('-','-')$ in the input file. We treat the first column and the first row as indices of the table formed by numeric values in the file. The value of entry (a, b) of the table $(a, b \in \{AA, AC, AG, AU, CA, CC, CG, CU, GA, GC, GG, GU, UA, UC, UG, UU, DD, BB\})$ is the cost of substituting a with b if $a \neq DD$, $a \neq BB$, $b \neq DD$ and $b \neq BB$; or the cost of deleting a if $a \neq DD$, $a \neq BB$ and $b = DD$; or the cost of inserting b if $a = DD$, $b \neq DD$ and $b \neq BB$; or the cost of breaking the bond of a if $a \neq DD$, $a \neq BB$ and $b = BB$; or the cost of breaking the bond of b if $a = BB$, $b \neq DD$ and $b \neq BB$. (Notice that the entries (DD, DD), (DD, BB), (BB, DD) and (BB, BB) have no meaning.)

The third input file is the RNA data file which contains the primary, secondary and tertiary structures information of two RNAs. We use the modified region table format where the energy column of secondary structure is removed.

Each RNA in the file contains three sections: first, the name of the RNA and the primary structure of the RNA, followed by the secondary structure of the RNA, followed by the tertiary structure of the RNA. Each section is separated by the '>' character alone on a line. If there is no tertiary structure, then after the '>' that ends

	AA	AC	AG	AU	CA	CC	CG	CU	GA	GC	GG	GU	UA	UC	UG	UU	DD	BB
AA	0	1	1	1	1	2	2	2	1	2	2	2	1	2	2	2	2	1
AC	1	0	1	1	2	1	2	2	2	1	2	2	2	1	2	2	2	1
AG	1	1	0	1	2	2	1	2	2	2	1	2	2	2	1	2	2	1
AU	1	1	1	0	2	2	2	1	2	2	2	1	2	2	2	1	2	1
CA	1	2	2	2	0	1	1	1	1	2	2	2	1	2	2	2	2	1
CC	2	1	2	2	1	0	1	1	2	1	2	2	2	1	2	2	2	1
CG	2	2	1	2	1	1	0	1	2	2	1	2	2	2	1	2	2	1
CU	2	2	2	1	1	1	1	0	2	2	2	1	2	2	2	1	2	1
GA	1	2	2	2	1	2	2	2	0	1	1	1	1	2	2	2	2	1
GC	2	1	2	2	2	1	2	2	1	0	1	1	2	1	2	2	2	1
GG	2	2	1	2	2	2	1	2	1	1	0	1	2	2	1	2	2	1
GU	2	2	2	1	2	2	2	1	1	1	1	0	2	2	2	1	2	1
UA	1	2	2	2	1	2	2	2	1	2	2	2	0	1	1	1	2	1
UC	2	1	2	2	2	1	2	2	2	1	2	2	1	0	1	1	2	1
UG	2	2	1	2	2	2	1	2	2	2	1	2	1	1	0	1	2	1
UU	2	2	2	1	2	2	2	1	2	2	2	1	1	1	1	0	2	1
DD	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	0	0
BB	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

Table 5.2: An example of score scheme for base pairs

secondary structure section, there should be another '>' in the next line.

The secondary and tertiary structures of the RNA are specified as follows:

(<stem number>) <start base> <end base> <stem size>

Figure 5.1 shows an example of an RNA input file which consists of the RNA *Alcaligenes eutrophus* and the RNA *Streptomyces bikiniensis.* We will use this data later.

Notice that we use traditional stems (i.e. stacked base pairs of maximal length) in the RNA input file as input, while we use maximal extended stems as input in algorithms discussed in Chapter 4. Our package has a procedure that groups traditional stems into maximal extended stems.

5.2 Experiment Results

In this section, we give experiment results of RNA tertiary structure alignment by the algorithms presented in Chapter 4. We performed extensive experiments of our alignment algorithm on real RNA structures. Experimental tests show that our algorithm can be used to compute alignment between RNA tertiary structures in practical applications. We also compare our results to Möhl *et al.*'s results and Wang and Zhang's results.

5.2.1 Results of Filtering Crossing Stem Pairs

We first give the results of filtering crossing stem pairs in the preprocessing crossing stem pairs step discussed in Section 4.5.2.

The RNA structures which we use are several tmRNAs that were used in Möhl *et al.*'s experiments [13]. These RNAs are selected from the tmRNA database [22]. They are the longest sequence (*Mycobacteriophage Bxz1*, MB), the shortest sequence (*Cyanidium caldarium*, CC) and the sequence that contains the largest crossing stems

```
Alcaligenes-eutrophus-pb-b
      1  AAAGCAGGCC AGGCAACCGC UGCCUGCACC GCAAGGUGCA GGGGGAGGAA
     51  AGUCCGGACU CCACAGGGCA GGGUGUUGGC UAACAGCCAU CCACGGCAAC
    101  GUGCGGAAUA GGGCCACAGA GACGAGUCUU GCCGCCGGGU UCGCCCGGCG
    151  GGAAGGGUGA AACGCGGUAA CCUCCACCUG GAGCAAUCCC AAAUAGGCAG
    201  GCGAUGAAGC GGCCCGCUGA GUCUGCGGGU AGGGAGCUGG AGCCGGCUGG
    251  UAACAGCCGG CCUAGAGGAA UGGUUGUCAC GCACCGUUUG CCGCAAGGCG
    301  GGCGGGGCGC ACAGAAUCCG GCUUAUCGGC CUGCUUUGCU U
>
 (    1)        1     337       10
 (    2)       11     326        1
 (    3)       12     278        7
 (    4)       20      45        2
 (    5)       23      42        8
 (    6)       59     183        4
 (    7)       71     179        5
 (    8)       77      89        4
 (    9)       91     105        1
 (   10)       92     103        4
 (   11)      106     174        2
 (   12)      111     172        2
 (   13)      127     156        4
 (   14)      132     151        8
 (   15)      187     235        4
 (   16)      197     226        6
 (   17)      206     220        5
 (   18)      242     261        8
 (   19)      281     308        2
 (   20)      284     305        9
>
 (    1)       50     324        3
 (    2)       54     321        5
 (    3)       66     215        4
>

Streptomyces-bikiniensis-gpb-h
      1  CGAGCCGGGC GGGCGGCCGC GUGGGGGUCU UCGGACCUCC CCGAGGAACG
     51  UCCGGGCUCC ACAGAGCAGG GUGGUGGCUA ACGGCCACCC GGGGUGACCC
    101  GCGGGACAGU GCCACAGAAA ACAGACCGCC GGGGACCUCG GUCCUCGGUA
    151  AGGGUGAAAC GGUGGUGUAA GAGACCACCA GCGCCUGAGG CGACUCAGGC
    201  GGCUAGGUAA ACCCCACUCG GAGCAAGGUC AAGAGGGGAC ACCCCGGUGU
    251  CCCUGCGCGG AUGUUCGAGG GCUGCUCGCC CGAGUCCGCG GGUAGACCGC
    301  ACGAGGCCGG CGGCAACGCC GGCCCUAGAU GGAUGGCCGU CGCCCCGACG
    351  ACCGCGAGGU CCCGGGGACA GAACCCGGCG UACAGCCCGA CUCGUCUG
>
 (    1)        1     394       10
 (    2)       11     383        1
 (    3)       12     341        7
 (    4)       20      43        2
 (    5)       23      40        7
 (    6)       57     223        4
 (    7)       69     219        5
 (    8)       74      88        5
 (    9)       89     103        1
 (   10)       90     101        4
 (   11)      104     214        2
 (   12)      109     212        2
 (   13)      126     153        2
 (   14)      128     149        9
 (   15)      161     179        6
 (   16)      182     201        8
 (   17)      227     298        4
 (   18)      235     254        8
 (   19)      257     289        6
 (   20)      267     283        6
 (   21)      305     324        8
 (   22)      343     367        5
 (   23)      350     361        4
>
 (    1)       48     381        3
 (    2)       52     378        5
 (    3)       64     277        4
>
```

Figure 5.1: RNA input file of *Alcaligenes eutrophus* and *Streptomyces bikiniensis*

(*Ureaplasma parvum*, UP). The region table representations for these RNA structures are shown in Figure 5.2 to 5.4, respectively.

```
Mycobacteriophage Bxz1 pre-tmRNA
       1  GGGCCUGACA AGGUUUCGAC UGGUCGAUGG ACAACUGAAC AGCGGGCGAG
      51  UGUUGGCCGC ACUUCUACUC UGAGUGAACG CGGCAACUGA UAAACGCAAC
     101  CGACACGGAU GCAACGGUGA CCGACGCCGA GAUCGAGGCC UUCUUUGCUG
     151  AAGAGGCUGC CGCUCUCGUC UGAAGGAACC AGCCUGGCUC AGCGUGCUGC
     201  UGUGCAGCGG CCAGGCUUCA UCUCUAACAG CAGCGAACGG ACAUGAGGGA
     251  GCGCAAACCC UCGUCCCAAA CAUCAUGAAU GCGUCGCACG GGCUCCAGCG
     301  UCAGGGGCCA GAGGUGGGAA ACGGUGUGAA ACUCCUGUCC UGGGGAUCAC
     351  CGACCGAUAC GCCAAACCAG GACUACGCCC GUAGAACGCA GUGAGAAAGA
     401  CACCAGGACA GGGGUUCGAG UCCCCUCAGG UCCACgu
>
 (    1)        1     433       7
 (    2)       20     406       5
 (    3)       34     392       4
 (    4)       43     381       5
 (    5)       56      84       5
 (    6)       63      78       3
 (    7)      146     194       6
 (    8)      154     185       6
 (    9)      322     356       4
 (   10)      326     350       4
 (   11)      333     346       3
 (   12)      410     426       5
>
 (    1)       48      70       4
 (    2)      337     373       7
>
```

Figure 5.2: Region table representation for MB

```
Cyanidium caldarium plastid pre-tmRNA
       1  GGGGCUGAAA GGAUAUUCGA CAUAUUAAUU UCGUGCGCUA UGAUGCAAGC
      51  CGAGAAUGCU UAUCUCGUAA AAAAGCAGAC AAAGAAAUAA AUGCAAACAA
     101  UAUUAUUGAA AUUAGCAAUA UUAGAAAACC AGCUCUAGUA GUCUAGCCUG
     151  AUUCAGUUAU UUCUAAAUUA UUUAUGUUAU GUUAUUUAAG CUUGUAGUAA
     201  CUAUCUAGUG UACAAUUUCU AUGGACGUGG GUUCAAUUCC CACCAGCUCC
     251  ACaa
>
 (    1)        1     250       7
 (    2)       21     223       4
 (    3)       27     217       4
 (    4)       33     213       8
 (    5)       42     205       4
 (    6)       46     194       5
 (    7)       51      67       5
 (    8)      227     243       5
>
 (    1)       57      77       5
>
```

Figure 5.3: Region table representation for CC

The results are shown in Table 5.3. The second column shows the original number of crossing stem pairs in ST_{CR}^{all}. The third column shows the number of crossing stem pairs in ST_{CR} after the first filter (preprocessing crossing inner local *m_stem* pairs). The fourth column shows the number of crossing stem pairs in ST_{CR} after the second filter (preprocessing crossing outer local *m_stem* pairs). The last column shows the total size reduction factor.

```
Ureaplasma parvum tmRNA
      1  GGGGAUGUCA CGGUUUCGAC GUGACACAUU AAUUUUUAAU UGCAGUGGGG
     51  UUAGCCCCUU AUCGCUUUCG AGGCAUUUUA AAUGCAGAAA AUAAAAAAUC
    101  UUCUGAAGUA GAAUUAAACC CAGCGUUUAU GGCUUCAGCU ACUAAUGCAA
    151  ACUACGCUUU UGCGUACUAA UUAGUUAUUA GUAGAAACGU UCAUUAACAU
    201  AAUUACUAUU GGUUGGUUUU UGGGCUUAUU UUACAAUAGU UUUAAAUUUA
    251  AAAUUCUUAU UUGUUGUUUA AAUUUAAAUA GAUUUAACAA AUAGUUAGUU
    301  AAUUUUAAAU UUGUUUUAUU AGUUAUUAAC UACACUAUUU UUAAUAAAAC
    351  UAAACUGUAG AUAUUAUUAA UUAUGUGUUG CGGAAAGGGG UUCGACUCCC
    401  CUCAUCUCCA CCA
>
 (   1)      1    409      7
 (   2)     20    382     10
 (   3)     31    372      4
 (   4)     37    366      4
 (   5)     43    357      4
 (   6)     47     59      4
 (   7)    158    186      6
 (   8)    189    225      5
 (   9)    196    218      3
 (  10)    200    215      4
 (  11)    241    278     12
 (  12)    296    332      8
 (  13)    308    324      4
 (  14)    386    402      5
>
 (   1)     53     66      3
 (   2)    205    240      7
 (   3)    257    294     10
 (   4)    313    350      8
>
```

Figure 5.4: Region table representation for UP

Aligned RNAs	$\lvert ST_{CR}^{all} \rvert$	$\lvert ST_{CR} \rvert$ after 1st filter	$\lvert ST_{CR} \rvert$ after 2nd filter	total size reduction
UP / UP	10193	3433	2345	77%
UP / MB	3617	1315	894	75%
UP / CC	1110	452	336	70%
MB / MB	1560	633	394	75%
MB / CC	325	164	114	65%
CC / CC	125	69	49	61%

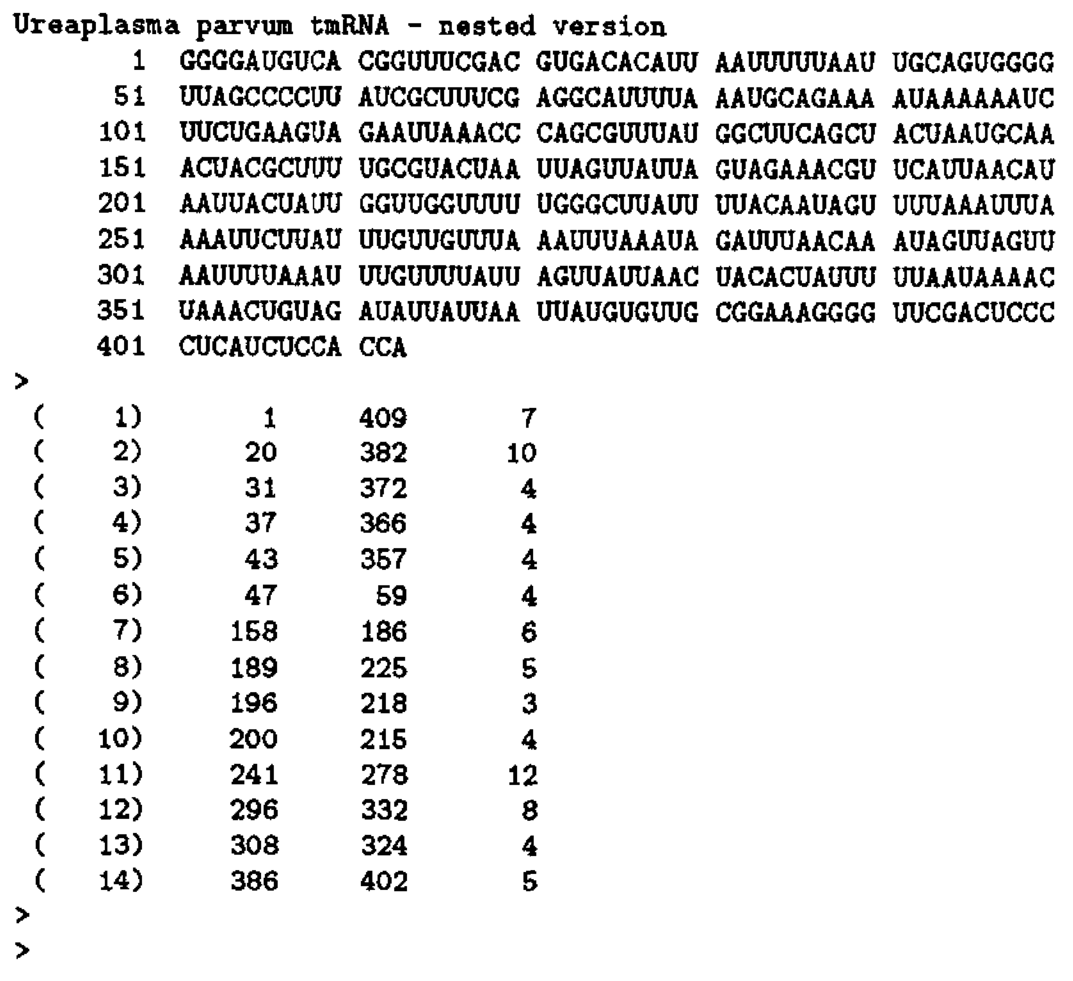
Table 5.3: The results of filtering crossing stem pairs

From the above results, we can see that many crossing stem pairs in ST_{CR}^{all} will not be realized in practice. The filtered crossing stem pairs set ST_{CR} is usually much smaller than the original ST_{CR}^{all}.

5.2.2 Comparison to Möhl *et al.*'s results

In this section, we compare our experiment results of the optimal alignment algorithm presented in Section 4.5 to Möhl *et al.*'s results [13].

The RNA structures which we use are MB, CC and UP introduced in the previous section, and a nested version (UPnest) of UP where all left crossing arcs are removed. The region table representation for UPnest is shown in Figure 5.5.



```
Ureaplasma parvum tmRNA - nested version
       1  GGGGAUGUCA CGGUUUCGAC GUGACACAUU AAUUUUUAAU UGCAGUGGGG
      51  UUAGCCCCUU AUCGCUUUCG AGGCAUUUUA AAUGCAGAAA AUAAAAAAUC
     101  UUCUGAAGUA GAAUUAAACC CAGCGUUUAU GGCUUCAGCU ACUAAUGCAA
     151  ACUACGCUUU UGCGUACUAA UUAGUUAUUA GUAGAAACGU UCAUUAACAU
     201  AAUUACUAUU GGUUGGUUUU UGGGCUUAUU UUACAAUAGU UUUAAAUUUA
     251  AAAUUCUUAU UUGUUGUUUA AAUUUAAAUA GAUUUAACAA AUAGUUAGUU
     301  AAUUUUAAAU UUGUUUUAUU AGUUAUUAAC UACACUAUUU UUAAUAAAAC
     351  UAAACUGUAG AUAUUAUUAA UUAUGUGUUG CGGAAAGGGG UUCGACUCCC
     401  CUCAUCUCCA CCA
>
 (    1)        1     409       7
 (    2)       20     382      10
 (    3)       31     372       4
 (    4)       37     366       4
 (    5)       43     357       4
 (    6)       47      59       4
 (    7)      158     186       6
 (    8)      189     225       5
 (    9)      196     218       3
 (   10)      200     215       4
 (   11)      241     278      12
 (   12)      296     332       8
 (   13)      308     324       4
 (   14)      386     402       5
>
>
```

Figure 5.5: Region table representation for UPnest

Comparing our experiment results with Möhl *et al.*'s results [13], we can find out that our implementation is much faster and uses less space. The results are shown in Table 5.4 (n = sequence length, s = maximal number of arcs in crossing stem, pk = number of pseudoknots, fixed parameter k = number of crossing maximal stem matches that overlap in a common point). Notice that we ran experiments on an Intel P8600 processor with 2.4 GHz, and Möhl *et al.* ran experiments on an Intel Xeon 5160 processor with 3.0 GHz. So our machine is slower. Our package is implemented in C and Möhl *et al.*'s program is implemented in C++. Thus the difference between two program's runtime should be due to algorithms.

					Möhl *et al.*'s results [13]		Our results	
Aligned RNAs	n	s	k	pk	runtime	memory	runtime	memory
UP / UP	413/413	10/10	1	4/4	726m 52s	$\leq$ 2 GB	207s	17,520 KB
UP / MB	413/437	10/7	1	4/2	172m 53s	$\leq$ 1 GB	51s	10,756 KB
UP / CC	413/254	10/5	1	4/1	11m 51s	$\leq$ 1 GB	21s	5,404 KB
UP / UPnest	413/413	10/0	0	4/0	4m 43s	$\leq$ 1 GB	11s	7,392 KB
MB / MB	437/437	7/7	1	2/2	43m 20s	$\leq$ 1 GB	20s	9,100 KB
MB / CC	437/254	7/5	1	2/1	3m 56s	$\leq$ 1 GB	8s	5,056 KB
MB / UPnest	437/413	7/0	0	2/0	3m 27s	$\leq$ 1 GB	8s	7,728 KB
CC/ CC	254/254	5/5	1	1/1	1m 11s	$\leq$ 1 GB	5s	3,028 KB
CC / UPnest	254/413	5/0	0	1/0	2m 6s	$\leq$ 1 GB	6s	4,712 KB
UPnest / UPnest	413/413	0/0	0	0/0	4m 21s	$\leq$ 1 GB	10s	7,296 KB

Table 5.4: Comparisons of our results to Möhl *et al.*'s results [13]

Notice that all alignments produced here are optimal alignments.

Möhl *et al.* did not present concrete optimal alignments in [13]. We show several representative alignments produced by our package in Figure 5.6 to 5.11, respectively. In the alignment produced by our package, paired bases which belong to secondary structure are indicated by round brackets at corresponding positions above (for the first RNA) or below (for the second RNA) them. Paired bases which belong to tertiary structure are indicated by round brackets at corresponding positions above (for the first RNA) or below (for the second RNA) them. A left bracket indicates a $5'$ end and a right bracket indicates a $3'$ end.

Table 5.4 shows that even though the worst case time and space complexities of our algorithm are the same as Möhl *et al.*'s, our algorithm could use much less resources (time and space) in practice to compute optimal alignment between two simple RNA tertiary structures.

5.2.3 Comparison to Wang and Zhang's results

We compare our experiment results of the constrained alignment algorithm presented in Section 4.6 to Wang and Zhang's constrained alignment results.

```
sequence 1: Ureaplasma parvum tmRNA
sequence 2: Mycobacteriophage Bxz1 pre-tmRNA

The optimal alignment score is 314.000
The optimal alignment is:

      (((((((            (--((-(((--(((( ((((  ((((  (-((((((-(  ---[[[-)-----))) - -
   1  GGGGAUGUCACGGUUUCGAC--GU-GAC--ACAUUAAUUUUUAAUUGC-AGUGGG-GUU---AGC-C-----CCUU-A-U
   1  GGGCCUGACAAGGUUUCGACUGGUCGAUGGACA--ACUG---AACAGCGGGCGAGUGUUGGCCGCACUUCUACUCUGAGU
      (((((((            (((((         --((((---     ((((([[[     (((((  ((( ]]]]

      --- ]]]                    ---    ---        -              -               -- -
  81  ---CGCUUUCGAGGCAUUUUAAAUGCA---GAAA---AUAAAAAA-UCUUCUGAAGUAGA-AUUAAACCCAGCGUU--U-
  81  GAACGCGG-CAA--C-UGAUAAACGCAACCGACACGGAUGCAACGGUGACC-GACGCCGAGAUCGAGGCCUUCUUUGCUG
      ))) ))))-)  -- -                                   -                       (((((

         - -              --   --     - --  ((((((  - - -            - ))))))  (-((((
 161  AUG-G-CUUCAGCUACUAAU--GCA--AACUA-C--GCUUUUGCGU-A-C-UAAUUAGUUAUU-AGUAGAAACG-UUCAU
 161  AAGAGGCUGCCGCU-CUCGUCUGAAGGAACCAGCCUG-----GC-UCAGCGUGCU--GCUGUGCAGCGGCCAGGCUUCAU
      (  ((((((     -               )))))) -----  -))))))    --

      - ((( (((( [-[[[[[[-)))-))))  ))-)))        ]]]---]]]](((((((((((--(    [[---[[[
 241  -UAACAUAAUUA-CUAUUG-GUU-GGUUUUUG-GGCUUAUUUUACAA---UAGUUUUAAAUUUAA--AAUUCUU---AUU
 241  CU--C-UAA-CAGC-A--GCGAACGGACAU-GAGG-G-AGCGCA-AACCCUCGUCCCAAACAUCAUGAAUGCGUCGCACG
        -- -   -    - --            -    - -      -

      [[[[[)))))))))))) ----     ]]]]]]]]]] ((((-((((    (-((( [[[[[[[[))))))))---))))
 321  UGUUGUUUAAAUUUAAAU----AGAUUUAACAAAUAGUUAGU-UAAUUUUAA-AUUUGUUUUAUUAGUUAUUA---ACUA
 321  -G--GCUCCAGCGUCAGGGGCCAGAGGUGGGAAA----CGGUGUGA----AACUCCUGUCCUGG-GGA--UCACCGACCG
      - --                              ----((((((((----   ((( [[[[[[[-)))--))))  ))))

                ]-]]]]]]]   )))-)     ))))  ))))-))))--)))))--)   (((((       ))))))))
 401  CACUAUUUUUA-AUAAAACUAAACU-GUAGAUAUUAUUAAUU-AUGU--GUUGC--GGAAAGGGGUUCGACUCCCCUCAU
 401  -A-UACGCCAAACCAGGACUACGCCCGUAGA-A---CGCAGUGA-GAAAGACACCAGGACAGGGGUUCGAGUCCCCUCAG
      - -         ]]]]]]]   )))))    - ---  ))))  -       )))))   (((((       ))))))))

      ))))
 481  CUCCACCA
 481  GUCCACgu
      ))))
```

Figure 5.6: Optimal alignment between UP and MB

```
sequence 1: Ureaplasma parvum tmRNA
sequence 2: Cyanidium caldarium plastid pre-tmRNA

The optimal alignment score is 266.000
The optimal alignment is:

      (((((((      - -     (((((((((( ((((  (((( - (-(-((((((  [[[)))     ]]]
   1  GGGGAUGUCACGG-U-UUCGACGUGACACAUUAAUUUUUAAUU-GC-A-GUGGGGUUAGCCCCUUAUCGCUUUCGAGGCA
   1  GGGGCUGAAA-GGAUAUUCGAC---AUA--UUAAUUUC--GUGCGCUAUGA----U--GC----AAGC-----CGAGA-A
      (((((((   -          (---(((--  ((((  --(((((((( ((----(--((----((((-----(((((-

        -                       -
  81  UU-UUAAAUGCAGAAAAUAAAAAAUC-UUCUGAAGUAGAAUUAAACCCAGCGUUUAUGGCUUCAGCUACUAAUGCAAACU
  81  UGCUUAUCU-C-G----UAAAAAAGCAGACA-AAG-A-AAU-AAAUGCAA-----A---CA--A--UAUUAUUG-AAAUU
      [[[[[ )))-)-)----     ]]]]]    -   - -   -        ----- ---  -- --        -

          ((((((                 ))))))  (((((  ((( (((( [[[[[[[)))))))  )))))
 161  ACGCUUUUGCGUACUAAUUAGUUAUUAGUAGAAACGUUCAUUAACAUAAUUACUAUUGGUUGGUUUUUGGCUUAUUUUA
 161  A-GC------A-A-UA-UUAG--A--A--A---AC---CAG---C-----U-CUA--G-UAG---U-----CU-AGCCU-
       -  ------ - -  -    -- -- -- ---  ---   --- ----- -   -- -   --- -----  -     -

      ]]]]]]]((((((((((((    [[[[[[[[[[[)))))))))))      ]]]]]]]]]] ((((((((    (((( [
 241  CAAUAGUUUUAAAUUUAAAAUUCUUAUUUGUUGUUUAAAUUUAAAUAGAUUUAACAAAUAGUUAGUUAAUUUUAAAUUUG
 241  -----G-------------AUUC--A---GU--------------UA--UUU--CUAA-A-UUA------UUUA----UG
      ----- -------------    -- ---  --------------  --   --    - -   ------    ----

      [[[[[[[))))))))))))  -    -    ]]]]]]]]   )))-)     ))))  ))))))))))))))   ((-((
 321  UUUUAUUAGUUAUUAACUACA-CUAU-UUUUAAUAAAACUAAACU-GUAGAUAUUAUUAAUUAUGUGUUGCGGAAAG-GG
 321  UU--AU--GUUAUU---UA-AGCU-UGUAGUAA-----CUA-UCUAGU-G-UAC----AAUU-UCUAU---GGAC-GUGG
        --  --      ---  - )))-))      ----- ))-))))))-)-)))----))))-  )))---)   -((((

      (       )))-)))))))))
 401  GUUCGACUCCC-CUCAUCUCCACCA
 401  GUUCAAUUCCCAC-CAGCUCCACaa
      (       )))))-)))))))
```

Figure 5.7: Optimal alignment between UP and CC

```
sequence 1: Ureaplasma parvum tmRNA
sequence 2: Ureaplasma parvum tmRNA - nested version

The optimal alignment score is 28.000
The optimal alignment is:

      (((((((            (((((((((( ((((  ((((  ((((((((  [[[))))    ]]]
   1  GGGGAUGUCACGGUUUCGACGUGACACAUUAAUUUUUAAUUGCAGUGGGGUUAGCCCCUUAUCGCUUUCGAGGCAUUUUA
   1  GGGGAUGUCACGGUUUCGACGUGACACAUUAAUUUUUAAUUGCAGUGGGGUUAGCCCCUUAUCGCUUUCGAGGCAUUUUA
      (((((((            (((((((((( ((((  ((((  ((((((((     ))))

                                                                                 (((
  81  AAUGCAGAAAAUAAAAAAUCUUCUGAAGUAGAAUUAAACCCAGCGUUUAUGGCUUCAGCUACUAAUGCAAACUACGCUUU
  81  AAUGCAGAAAAUAAAAAAUCUUCUGAAGUAGAAUUAAACCCAGCGUUUAUGGCUUCAGCUACUAAUGCAAACUACGCUUU
                                                                                 (((

      (((                 ))))))  (((((  ((( (((( [[[[[[[)))))))  )))))        ]]]]]]]
 161  UGCGUACUAAUUAGUUAUUAGUAGAAACGUUCAUUAACAUAAUUACUAUUGGUUGGUUUUUGGGCUUAUUUUACAAUAGU
 161  UGCGUACUAAUUAGUUAUUAGUAGAAACGUUCAUUAACAUAAUUACUAUUGGUUGGUUUUUGGGCUUAUUUUACAAUAGU
      (((                 ))))))  (((((  ((( ((((        )))))))  )))))

      ((((((((((((    [[[[[[[[[[))))))))))))      ]]]]]]]]]] ((((((((    ((((  [[[[[[[
 241  UUUAAAUUUAAAAUUCUUAUUUGUUGUUUAAAUUUAAAUAGAUUUAACAAAUAGUUAGUUAAUUUUAAAUUUGUUUUAUU
 241  UUUAAAUUUAAAAUUCUUAUUUGUUGUUUAAAUUUAAAUAGAUUUAACAAAUAGUUAGUUAAUUUUAAAUUUGUUUUAUU
      ((((((((((((              ))))))))))))                 ((((((((    ((((

      ))))))))))))          ]]]]]]]    ))))     ))))  ))))))))))))))   (((((       )))
 321  AGUUAUUAACUACACUAUUUUUAAUAAAACUAAACUGUAGAUAUUAUUAAUUAUGUGUUGCGGAAAGGGGUUCGACUCCC
 321  AGUUAUUAACUACACUAUUUUUAAUAAAACUAAACUGUAGAUAUUAUUAAUUAUGUGUUGCGGAAAGGGGUUCGACUCCC
      ))))))))))))                     ))))     ))))  ))))))))))))))   (((((       )))

      )))))))))
 401  CUCAUCUCCACCA
 401  CUCAUCUCCACCA
      )))))))))
```

Figure 5.8: Optimal alignment between UP and UPnest

```
sequence 1: Mycobacteriophage Bxz1 pre-tmRNA
sequence 2: Cyanidium caldarium plastid pre-tmRNA

The optimal alignment score is 282.000
The optimal alignment is:

      (((((((      - -     (-((((   -- - -    (((--(     (((((-[[[[    (((((  ((( ]]]]
   1  GGGCCUGACAAGG-U-UUCGAC-UGGUCGA--U-G-GACAACU--GAACAGCGGGC-GAGUGUUGGCCGCACUUCUACUC
   1  GGGGCUGA-AAGGAUAUUCGACAU-AUUAAUUUCGUG-C-GCUAUGAU--GCAAGCCGAGAAU-----GC--UU-AUCUC
      ((((((( -            (((-(  ((((  (((-(-(((( (((--((((((((((  [-----[[--[[- ))))

      -     ))) )))))                                                             ((((
  81  -UGAGUGAACGCGGCAACUGAUAAACGCAACCGACACGGAUGCAACGGUGACCGACGCCGAGAUCGAGGCCUUCUUUGCU
  81  GUAAAA-AA-GC---A---GACAAA-GAAAU--A-A---AUGCAA-----AC--A-----A--U--A----UUAUU-G--
      )     -]]-]]---]---      -     -- - ---      -----  -- ----- -- -- ----     - --

      ((  ((((((                    ))))))   ))))))
 161  GAAGAGGCUGCCGCUCUCGUCUGAAGGAACCAGCCUGGCUCAGCGUGCUGCUGUGCAGCGGCCAGGCUUCAUCUCUAACA
 161  -AA-A---------U-UAG-C--AAU-A-----------------U--U-------AG-----A------A-----AAC-
      -  - --------- -   - --   - ----------------- -- -------  ----- ------ -----   -

 241  GCAGCGAACGGACAUGAGGGAGCGCAAACCCUCGUCCCAAACAUCAUGAAUGCGUCGCACGGGCUCCAGCGUCAGGGGCC
 241  -CAGCU--C-----U-AGU-AG---------UC-U---AG-C--C-UGA-U---UC--A-GU--U--AUU-UC-------
      -     -- ----- -   - ---------  - ---  - -- -   - ---  -- -  -- --   - -------

                  ((((((((   ((( [[[[[[[)))))))  ))))          ]]]]]]]   )))))   -
 321  AGAGGUGGGAAACGGUGUGAAACUCCUGUCCUGGGGAUCACCGACCGAUACGCCAAACCAGGACUACGCCCGUAG-AACG
 321  -----U---AAA---U-U--A--U--U-U-------AU----G----UUAUGUUAUU-------UAAGCUUGUAGUAACU
      ----- ---   --- - -- -- -- - -------  ---- ----          -------   )))))       )

      --)-))) -        ))))-)   ((-(((       )))-)))))))))
 401  --C-AGUG-AGAAAGACACCA-GGACAG-GGGUUCGAGUCCC-CUCAGGUCCACgu
 401  AUCUAGUGUACAAUUUC-U-AUGGAC-GUGGGUUCAAUUCCCAC-CAGCUCCACaa
      )))))))))))))))  -)-)))   -(((((       )))))-)))))))
```

Figure 5.9: Optimal alignment between MB and CC

```
sequence 1: Mycobacteriophage Bxz1 pre-tmRNA
sequence 2: Ureaplasma parvum tmRNA - nested version

The optimal alignment score is 304.500
The optimal alignment is:

      (((((((            (((((         --((((---     ((((([[[[    (((((  ((( ]]]]- -
   1  GGGCCUGACAAGGUUUCGACUGGUCGAUGGACA--ACUG---AACAGCGGGCGAGUGUUGGCCGCACUUCUACUC-U-GA
   1  GGGGAUGUCACGGUUUCGAC--GU-GAC--ACAUUAAUUUUUAAUUGC-AGU-GGGGUUAGC----C----CCUUAUCGC
      (((((((            (--((-(((--(((( ((((  ((((  (-(((-((((     ----)----)))

        ))) )))))                                      -                       ((((((
  81  GUGAACGCGGCAACUGAUAAACGCAACCGACACGGAUGCAACGGUGACC-GACGCCGAGAUCGAGGCCUUCUUUGCUGAA
  81  UUU--CGAGGCAUUU--UAAAUGCA---GAAA---AUAAAAAA-UCUUCUGAAGUAGA-AUUAAACCCAGCGUU--U-AU
         --          --         ---    ---        -              -               -- -

       ((((((     -     -          ))))----)) -  ))))))    --
 161  GAGGCUGCCGCU-CUCGU-CUGAAGGAACCAGC----CUG-GCUCAGCGUGCU--GCUGUGCAGCGGCCAGGCUUCAUCU
 161  G-G-CUUCAGCUACUAAUGCA-AACUA-C--GCUUUU--GCG-U-A-C-UAAUUAGUUAUU-AGUAGAAACG-UUCAU-U
       - -                 -     - --  ((((--(( - - - -            - ))))))  (-(((( -

      -- -   -    -                  -                                          -
 241  --C-UAA-CAGC-AGCGAACGGACAUGAGGG-AGCGCAAACCCUCGUCCCAAACAUCAUGAAUGCGUCGCACGG-GCUCC
 241  AACAUAAUUA-CUAUUGGUUGGUUUUU-GGGCUUAUUUUACAAUAGUUUUAAAUUUAA--AAUUCUUAUUU-GUUGUUUA
      ((( ((((  -      )))))))  )-))))               (((((((((((--(          -    ))))

                               (((-(-(((( ----  ((( [[[[[[[-)))---))))  ))-))        -
 321  AGCGUCAGGGGCCAGAGGUGGGAAACGG-U-GUGAA----ACUCCUGUCCUGG-GGA---UCACCGAC-CGAUACGCCA-
 321  AAUUUAAAU----AGAUUUAACAAAUAGUUAGUUAAUUUUAAAUUUGUUUUAUUAGUUAUUAACU-ACACUAUUUUUAAU
      )))))))) ----                ((((((((    ((((         )))))))))))-)

        ]]]]]]]   )))))    - ---  ))))  -       )))))   (((((       ))))))))))))
 401  AACCAGGACUACGCCCGUAGA-A---CGCAGUGA-GAAAGACACCAGGACAGGGGUUCGAGUCCCCUCAGGUCCACgu
 401  AA--A--ACUAAACU-GUAGAUAUUAUUAAUU-AUGU--GUUGC--GGAAAGGGGUUCGACUCCCCUCAUCUCCACCA
        -- --     )))-)     ))))  ))))-))))--)))))--)   (((((       ))))))))))))
```

Figure 5.10: Optimal alignment between MB and UPnest

```
sequence 1: Cyanidium caldarium plastid pre-tmRNA
sequence 2: Ureaplasma parvum tmRNA - nested version

The optimal alignment score is 250.500
The optimal alignment is:

      (((((((   -          (---(((--  ((((  --(((((((( ((----(--((----((--((---(((((-
   1  GGGGCUGAAA-GGAUAUUCGAC---AUA--UUAAUUUC--GUGCGCUAUGA----U--GC----AA--GC---CGAGA-A
   1  GGGGAUGUCACGG-U-UUCGACGUGACACAUUAAUUUUUAAUU-GC-A-GUGGGGUUAGCCCCUUAUCGCUUUCGAGGCA
      (((((((      - -     (((((((((( ((((  ((((  - (-(-((((((     ))))

      [[[[[ )))-)-)----     ]]]]]    -   - -   -        ----- ---  -- --        -
  81  UGCUUAUCU-C-G----UAAAAAAGCAGACA-AAG-A-AAU-AAAUGCAA-----A---CA--A--UAUUAUUG-AAAUU
  81  UU-UUAAAUGCAGAAAAUAAAAAAUC-UUCUGAAGUAGAAUUAAACCCAGCGUUUAUGGCUUCAGCUACUAAUGCAAACU
        -                       -

       -  ------ - -  -    -- -- -- ---  ---   --- ----- -       ------ -----  -   ---
 161  A-GC------A-A-UA-UUAG--A--A--A---AC---CAG---C-----U-CUAGUAG------U-----CU-AGC---
 161  ACGCUUUUGCGUACUAAUUAGUUAUUAGUAGAAACGUUCAUUAACAUAAUUACUAUUGGUUGGUUUUUGGGCUUAUUUUA
          ((((((                 ))))))  (((((  ((( ((((        )))))))  )))))

       -- - -------------    -- ---   -------------- --   --    - -   ------    ----
 241  C--U-G-------------AUUC--A---GUU--------------A--UUU--CUAA-A-UUA------UUUA----UG
 241  CAAUAGUUUUAAAUUUAAAAUUCUUAUUUGUUGUUUAAAUUUAAAUAGAUUUAACAAAUAGUUAGUUAAUUUUAAAUUUG
             ((((((((((((              ))))))))))))                 ((((((((    ((((

        --  --      ---  - )))-)) --     -- ))-))))))-)-)))----))))-  )))---)   -(((((
 321  UU--AU--GUUAUU---UA-AGCU-UGU--AGUAA--CUA-UCUAGU-G-UAC----AAUU-UCUAU---GGAC-GUGGG
 321  UUUUAUUAGUUAUUAACUACA-CUAUUUUAAUAAAACUAAACU-GUAGAUAUUAUUAAUUAUGUGUUGCGGAAAG-GGG
             ))))))))))))  -                   )))-)     ))))  ))))))))))))))   ((-(((

             )))))-)))))))
 401  UUCAAUUCCCAC-CAGCUCCACaa
 401  UUCGACUCCC-CUCAUCUCCACCA
             )))-)))))))))
```

Figure 5.11: Optimal alignment between CC and UPnest

The first group of RNAs which we use are the same as the RNAs used in previous section. We only perform experiments on UP, MB and CC. The results are shown in Table 5.5 (n = sequence length, s = maximal number of arcs in crossing stem, pk = number of pseudoknots, fixed parameter k = number of crossing maximal stem matches that overlap in a common point, (i, j) = the start bases's positions of the two selected maximal stems in our constrained alignment algorithm).

					Wang and Zhang's results		Our results		
Aligned RNAs	n	s	k	pk	runtime	memory	(i, j)	runtime	memory
UP / UP	413/413	10/10	1	4/4	8s	3,764 KB	(296, 296)	82s	11,416 KB
UP / MB	413/437	10/7	1	4/2	7s	4,020 KB	(296, 322)	10s	7,696 KB
UP / CC	413/254	10/5	1	4/1	4s	2,752 KB	(47, 51)	4s	4,568 KB
MB / MB	437/437	7/7	1	2/2	5s	4,180 KB	(322, 322)	3s	7,140 KB
MB / CC	437/254	7/5	1	2/1	3s	2,492 KB	(48, 51)	2s	4,908 KB
CC/ CC	254/254	5/5	1	1/1	3s	1,796 KB	(57, 57)	1s	2,832 KB

Table 5.5: First comparison of our results to Wang and Zhang's constrained alignment results

The concrete alignments produced by our algorithm and Wang and Zhang's algorithm are shown in Figure 5.12 to 5.14, respectively[1]. Notice that our constrained alignment algorithm is based on our optimal alignment algorithm which is exponential, and Wang and Zhang's constrained alignment algorithm is based on their optimal alignment algorithm which is polynomial. Thus it is reasonable that our program uses more time and space.

All alignments produced in this section are constrained alignments. They are not guaranteed to be optimal. For example, comparing Figure 5.13 to Figure 5.7, we can find that the constrained alignments between UP and CC produced by Wang and Zhang's algorithm and our algorithm are not optimal. But these two constrained alignments are near-optimal. Comparing Figure 5.12 to Figure 5.6, we can find that the constrained alignments between UP and MB produced by Wang and Zhang's algorithm and our algorithm are optimal. Similar for the constrained alignments between MB and CC.

[1]Notice that we do not include the alignment between UP and itself, the alignment between MB and itself and the alignment between CC and itself, since it is obvious that there are matches at all positions of these alignments.

```
---------------------------------------------------------------------------------------
by Wang and Zhang's algorithm:

Ureaplasma-parvum-tmRNA
Mycobacteriophage-Bxz1-pre-tmRNA

score = 314.000000

      (((((((            (((((((((( -(--(((  ((((  ((((--((((   ---[[[-)----)))- - - --
   1  GGGGAUGUCACGGUUUCGACGUGACACAUU-A--AUUUUUAAUUGCAGU--GGGGUU---AGC-C----CCU-U-A-U--
   1  GGGCCUGACAAGGUUUCGACUGGUCG-AUGGACAACU--GAACAGCGGGCGAGUGUUGGCCGCACUUCUACUCUGAGUGA
      (((((((            (((((  -          (((--(     ((((([[[   (((((  ((( ]]]     ))

      - ]]]                        ---    ---        -              -              -- -
  81  -CGCUUUCGAGGCAUUUUAAAUGCA---GAAA---AUAAAAAA-UCUUCUGAAGUAGA-AUUAAACCCAGCGUU--U-AU
  81  ACGCGG-CAA--C-UGAUAAACGCAACCGACACGGAUGCAACGGUGACC-GACGCCGAGAUCGAGGCCUUCUUUGCUGAA
      ) ))))-)  -- -                                    -                        ((((((

      -- -                  --   --     - --  ((((((  - - -             - ))))))  (-(((( -
 161  G-G-CUUCAGCUACUAAU--GCA--AACUA-C--GCUUUUGCGU-A-C-UAAUUAGUUAUU-AGUAGAAACG-UUCAU-U
 161  GAGGCUGCCGCU-CUCGUCUGAAGGAACCAGCCUG-----GC-UCAGCGUGCU--GCUGUGCAGCGGCCAGGCUUCAUCU
       ((((((     -                  )))))) -----  -))))))    --

      ((( (((( [-[[[[[[-)))-))))  ))-)))          ]]]---]]]]((((((((((--(    [[---[[[[[
 241  AACAUAAUUA-CUAUUG-GUU-GGUUUUUG-GGCUUAUUUUACAA---UAGUUUUAAAUUUAA--AAUUCUU---AUUUG
 241  --C-UAA-CAGC-A--GCGAACGGACAU-GAGG-G-AGCGCA-AACCCUCGUCCCAAACAUCAUGAAUGCGUCGCACG-G
      -- -   -    - --               -    - -    -                                 -

      [[[))))))))))))) ----     ]]]]]]]]]] ((((-((((    (-((( [[[[[[[[[))))))))---))))
 321  UUGUUUAAAUUUAAAU----AGAUUUAACAAAUAGUUAGU-UAAUUUUAA-AUUUGUUUUAUUAGUUAUUA---ACUACA
 321  --GCUCCAGCGUCAGGGGCCAGAGGUGGGAAA----CGGUGUGA----AACUCCUGUCCUGG-GGA--UCACCGACCG-A
      --                                ----((((((((----   ((( [[[[[[[-)))--))))  ))))-

               ]-]]]]]]]    -))))      ))))  ))))))))--)--)))))    (((((        ))))))))))
 401  CUAUUUUUA-AUAAAACUAA-ACUGUAGAUAUUAUUAAUUAUGU--G--UUGCGGAAAGGGGUUCGACUCCCCUCAUCUC
 401  -UACGCCAAACCAGGACUACGCCCGUAGA-ACGC--AGUGA-GAAAGACACCAGGACAGGGGUUCGAGUCCCCUCAGGUC
      -          ]]]]]]]  )))))    -   )--)))  -        )))))   (((((        ))))))))))

      )
 481  CACCA
 481  CACgu
      )

---------------------------------------------------------------------------------------
by our algorithm:

sequence 1: Ureaplasma parvum tmRNA
sequence 2: Mycobacteriophage Bxz1 pre-tmRNA
The constrained stem pair:
1st stem:(outermost arc, innermost arc), 2nd stem:(outermost arc, innermost arc)
          ((296, 332), (311, 321)),               ((322, 356), (335, 344))

The optimal constrained alignment score is 314.000
The optimal constrained alignment is:

      (((((((            (--((-(((--(((( ((((  ((((  (-((((((-(  ---[[[-)-----))) - -
   1  GGGGAUGUCACGGUUUCGAC--GU-GAC--ACAUUAAUUUUUAAUUGC-AGUGGG-GUU---AGC-C-----CCUU-A-U
   1  GGGCCUGACAAGGUUUCGACUGGUCGAUGGACA--ACUG---AACAGCGGGCGAGUGUUGGCCGCACUUCUACUCUGAGU
      (((((((            (((((            --((((---     ((((([[[[    (((((  ((( ]]]]

      --- ]]]                       ---     ---        -              -             -- -
  81  ---CGCUUUCGAGGCAUUUUAAAUGCA---GAAA---AUAAAAAA-UCUUCUGAAGUAGA-AUUAAACCCAGCGUU--U-
  81  GAACGCGG-CAA--C-UGAUAAACGCAACCGACACGGAUGCAACGGUGACC-GACGCCGAGAUCGAGGCCUUCUUUGCUG
      ))) ))))-)  -- -                                    -                        (((((

         - -                --   --     - --  ((((((  - - -             - ))))))  (-((((
 161  AUG-G-CUUCAGCUACUAAU--GCA--AACUA-C--GCUUUUGCGU-A-C-UAAUUAGUUAUU-AGUAGAAACG-UUCAU
 161  AAGAGGCUGCCGCU-CUCGUCUGAAGGAACCAGCCUG-----GC-UCAGCGUGCU--GCUGUGCAGCGGCCAGGCUUCAU
      (  ((((((     -                  )))))) -----  -))))))    --

      - ((( (((( [-[[[[[[-)))-))))  ))-)))          ]]]---]]]]((((((((((--(    [[---[[[
 241  -UAACAUAAUUA-CUAUUG-GUU-GGUUUUUG-GGCUUAUUUUACAA---UAGUUUUAAAUUUAA--AAUUCUU---AUU
 241  CU--C-UAA-CAGC-A--GCGAACGGACAU-GAGG-G-AGCGCA-AACCCUCGUCCCAAACAUCAUGAAUGCGUCGCACG
        -- -   -    - --               -    - -    -

      [[[[[)))))))))))) ----     ]]]]]]]]]] ((((-((((    (-((( [[[[[[[[[))))))))---))))
 321  UGUUGUUUAAAUUUAAAU----AGAUUUAACAAAUAGUUAGU-UAAUUUUAA-AUUUGUUUUAUUAGUUAUUA---ACUA
 321  -G--GCUCCAGCGUCAGGGGCCAGAGGUGGGAAA----CGGUGUGA----AACUCCUGUCCUGG-GGA--UCACCGACCG
      - --                                ----((((((((----   ((( [[[[[[[-)))--))))  ))))

                ]-]]]]]]]   )))-)      ))))  ))))-))))--)))))--)    (((((       ))))))))
 401  CACUAUUUUUA-AUAAAACUAAACU-GUAGAUAUUAUUAAUU-AUGU--GUUGC--GGAAAGGGGUUCGACUCCCCUCAU
 401  -A-UACGCCAAACCAGGACUACGCCCGUAGA-A---CGCAGUGA-GAAAGACACCAGGACAGGGGUUCGAGUCCCCUCAG
      - -          ]]]]]]]  )))))    - ---  ))))  -        )))))   (((((       ))))))))

      ))))
 481  CUCCACCA
 481  GUCCACgu
      ))))
```

Figure 5.12: Constrained alignment between UP and MB

```
-----------------------------------------------------------------------------------------
by Wang and Zhang's algorithm:

Ureaplasma-parvum-tmRNA
Cyanidium-caldarium-plastid-pre-tmRNA

score = 267.000000

     (((((((      - -     (((((((((( (((( -- ----((--((  ((-((((((  [[[----))))    --
  1  GGGGAUGUCACGG-U-UUCGACGUGACACAUUAAUUU--U----UA--AUUGCA-GUGGGGUUAGC----CCCUUAUC--
  1  GGGGCUGAAA-GGAUAUUCGAC---AUA--UUAAUUUCGUGCGCUAUGAU-GCAAGCCGAGAAUGCUUAUCUCGUAAAAA
     (((((((   -          (---(((--  ((((  (((((((( (((-((((((((((  [[[[[ )))))     ]

     -]]]
 81  -GCUUUCGAGGCAUUUUAAAUGCAGAAAAUAAAAAAUCUUCUGAAGUAGAAUUAAACCCAGCGUUUAUGGCUUCAGCUAC
 81  AGCA---GA--CAAAG-AAAU--A-AA--UGCAAA--C-----AA-UAUUAUUGAA---A---UU-A-G-CA--A--UAU
     ]]]]---  --     -     -- -  --      -- -----  -          --- ---  - - -  -- --

                    ((((((                 ))))))  (((((  ((( (((( [[[[[[[)))))))  ))
161  UAAUGCAAACUACGCUUUUGCGUACUAAUUAGUUAUUAGUAGAAACGUUCAUUAACAUAAUUACUAUUGGUUGGUUUUUG
161  UA--GAAAACCA-GC-------U-CUAGU-AGUC-U-AG------C---C-UG---AU---U-C-A--G-------UU--
       --        -  ------- -     -    - -  ------ --- -  ---  --- - - -- -------  --

     )))        ]]]]]]](((((((((((((    [[[[[[[[[[[)))))))))))       ]]]]]]]]] (((((((
241  GGCUUAUUUUACAAUAGUUUUAAAUUUAAAAUUCUUAUUUGUUGUUUAAAUUUAAAUAGAUUUAACAAAUAGUUAGUUAA
241  -----AUUU--C--UA-----A--------AUU---A-------------------U---UU-A--------U--G----
     -----    -- --  ----- --------   --- ------------------- ---  - -------- -- ----

     (    (((( [[[[[[[[))))))))))))          ]]]]]]]]   ))-))   - -- ))-))- - --)))))
321  UUUUAAAUUUGUUUUAUUAGUUAUUAACUACACUAUUUUUAAUAAAACUAAAC-UGUAG-A--UAU-UA-U-U--AAUUA
321  -UU-A----UGU---------------------UAUUU-----------AAGCUUGUAGUAACUAUCUAGUGUACAAUU-
     -  - ----   ---------------------     -----------  )))))        )))))))))))))))-

     )))))))))   ((-(((       )))-)))))))))
401  UGUGUUGCGGAAAG-GGGUUCGACUCCC-CUCAUCUCCACCA
401  UCUAU---GGAC-GUGGGUUCAAUUCCCAC-CAGCUCCACaa
       )))---)   -(((((       )))))-)))))))

-----------------------------------------------------------------------------------------
by our algorithm:

sequence 1: Ureaplasma parvum tmRNA
sequence 2: Cyanidium caldarium plastid pre-tmRNA
The constrained stem pair:
1st stem:(outermost arc, innermost arc), 2nd stem:(outermost arc, innermost arc)
          (( 47,  59), ( 50,  56)),                (( 51,  67), ( 55,  63))

The optimal constrained alignment score is 267.500
The optimal constrained alignment is:

     (((((((      - -     (((((((((( (((( -- ----((--((  (-(((((-((   [[[---))-))   -
  1  GGGGAUGUCACGG-U-UUCGACGUGACACAUUAAUUU--U----UA--AUUGC-AGUGG-GGUUAGC---CC-CUUAUC-
  1  GGGGCUGAAA-GGAUAUUCGAC---AUA--UUAAUUUCGUGCGCUAUGAU-GCAAGCCGAGAA-UGCUUAUCUCGUAAAA
     (((((((   -          (---(((--  ((((  (((((((( (((-((((((((((  -[[[[[ )))))

     --]]]
 81  --GCUUUCGAGGCAUUUUAAAUGCAGAAAAUAAAAAAUCUUCUGAAGUAGAAUUAAACCCAGCGUUUAUGGCUUCAGCUA
 81  AAGCA---GA--CAAAG-AAAU--A-AA--UGCAAA--C-----AA-UAUUAUUGAA---A---UU-A-G-CA--A--UA
     ]]]]]---  --     -     -- -  --      -- -----  -          --- ---  - - -  -- --

                     ((((((                 ))))))  (((((  ((( (((( [[[[[[[)))))))  )
161  CUAAUGCAAACUACGCUUUUGCGUACUAAUUAGUUAUUAGUAGAAACGUUCAUUAACAUAAUUACUAUUGGUUGGUUUUU
161  UUA--GAAAACCA-GC-------U-CUAGU-AGUC-U-AG------C---C-UG---AU---U-C-A--G-------UU-
        --        -  ------- -     -    - -  ------ --- -  ---  --- - - -- -------  -

     ))))        ]]]]]]](((((((((((((    [[[[[[[[[[[)))))))))))       ]]]]]]]]] ((((((
241  GGGCUUAUUUUACAAUAGUUUUAAAUUUAAAAUUCUUAUUUGUUGUUUAAAUUUAAAUAGAUUUAACAAAUAGUUAGUUA
241  ------AUUU--C--UA-----A--------AUU---A-------------------U---UU-A--------U--G---
     ------    -- --  ----- --------   --- ------------------- ---  - -------- -- ---

     ((    (((( [[[[[[[[))))))))))))          ]]]]]]]]   )))-)   - -- ))-))- - --))))
321  AUUUUAAAUUUGUUUUAUUAGUUAUUAACUACACUAUUUUUAAUAAAACUAAACU-GUAG-A--UAU-UA-U-U--AAUU
321  --UU-A----UGU---------------------UAUUU-----------AAGCUUGUAGUAACUAUCUAGUGUACAAUU
     --  - ----   ---------------------     -----------  )))))       ))))))))))))))))

     ))))))))))   ((-(((       )))-)))))))))
401  AUGUGUUGCGGAAAG-GGGUUCGACUCCC-CUCAUCUCCACCA
401  -UCUAU---GGAC-GUGGGUUCAAUUCCCAC-CAGCUCCACaa
     -  )))---)   -(((((       )))))-)))))))
```

Figure 5.13: Constrained alignment between UP and CC

```
--------------------------------------------------------------------------------------
by Wang and Zhang's algorithm:

Mycobacteriophage-Bxz1-pre-tmRNA
Cyanidium-caldarium-plastid-pre-tmRNA

score = 282.000000

     (((((((      - -     (((((   -- - -    (((--(     (((((-[[[[    (((((   ((( ]]]-
  1  GGGCCUGACAAGG-U-UUCGACUGGUCGA--U-G-GACAACU--GAACAGCGGGC-GAGUGUUGGCCGCACUUCUACUC-
  1  GGGGCUGA-AAGGAUAUUCGACAUAUUAAUUUCGUG-C-GCUAUGAU--GCAAGCCGAGAAU-----GC--UU-AUCUCG
     ((((((( -            ((((  ((((  (((-(-(((( (((--((((((((((( [-----[[--[[- )))))

          ))) )))))                                                             (((((
 81  UGAGUGAACGCGGCAACUGAUAAACGCAACCGACACGGAUGCAACGGUGACCGACGCCGAGAUCGAGGCCUUCUUUGCUG
 81  UAAAA-AA-GC---A---GACAAA-GAAAU--A-A---AUGCAA-----AC--A-----A--U--A----UUAUU-G---
          -]]-]]---]---      -     -- - ---      -----  -- ----- -- -- ----     - ---

     (  ((((((                    ))))))   ))))))
161  AAGAGGCUGCCGCUCUCGUCUGAAGGAACCAGCCUGGCUCAGCGUGCUGCUGUGCAGCGGCCAGGCUUCAUCUCUAACAG
161  AA-A---------U-UAG-C--AAU-A-----------------U--U-------AG-----A------A-----AAC--
       - ---------- -   - --   - ----------------- -- -------  ----- ------ -----   --


     CAGCGAACGGACAUGAGGGAGCGCAAACCCUCGUCCCAAACAUCAUGAAUGCGUCGCACGGGCUCCAGCGUCAGGGGCCA
241  CAGCU--C-----U-AGU-AG---------UC-U---AG-C--C-UGA-U---UC--A-GU--U--AUU-UC--------
          -- ----- -   -  ---------  - ---  - -- -   - ---  -- -  -- --   -  --------

                ((((((((   ((( [[[[[[[)))))))  ))))          ]]]]]]]   )))))    -   -
321  GAGGUGGGAAACGGUGUGAAACUCCUGUCCUGGGGAUCACCGACCGAUACGCCAAACCAGGACUACGCCCGUAG-AACG-
321  ----U---AAA---U-U--A--U--U-U-------AU----G----UUAUGUUAUU-------UAAGCUUGUAGUAACUA
     ---- ---   --- - -- -- -- - -------  ---- ----          -------   )))))       ))

     -)-))) -        )))))   ((-(((       )))-)))))))))
401  -C-AGUG-AGAAAGACACCAGGACAG-GGGUUCGAGUCCC-CUCAGGUCCACgu
401  UCUAGUGUACAAUUUC-UAUGGAC-GUGGGUUCAAUUCCCAC-CAGCUCCACaa
     )))))))))))))) -))))   -(((((       )))))-)))))))

--------------------------------------------------------------------------------------
by our algorithm:

sequence 1: Mycobacteriophage Bxz1 pre-tmRNA
sequence 2: Cyanidium caldarium plastid pre-tmRNA
The constrained stem pair:
1st stem:(outermost arc, innermost arc), 2nd stem:(outermost arc, innermost arc)
         (( 48,  70), ( 51,  67)),               (( 51,  67), ( 55,  63))

The optimal constrained alignment score is 282.000
The optimal constrained alignment is:

     (((((((      - -     (-((((   -- - -    (((--(     (((((-[[[[    (((((  ((( ]]]]
  1  GGGCCUGACAAGG-U-UUCGAC-UGGUCGA--U-G-GACAACU--GAACAGCGGGC-GAGUGUUGGCCGCACUUCUACUC
  1  GGGGCUGA-AAGGAUAUUCGACAU-AUUAAUUUCGUG-C-GCUAUGAU--GCAAGCCGAGAAU-----GC--UU-AUCUC
     ((((((( -            (((-(  ((((  (((-(-(((( (((--((((((((((( [-----[[--[[- ))))

     -     ))) )))))                                                             ((((
 81  -UGAGUGAACGCGGCAACUGAUAAACGCAACCGACACGGAUGCAACGGUGACCGACGCCGAGAUCGAGGCCUUCUUUGCU
 81  GUAAAA-AA-GC---A---GACAAA-GAAAU--A-A---AUGCAA-----AC--A-----A--U--A----UUAUU-G--
     )     -]]-]]---]---      -     -- - ---      -----  -- ----- -- -- ----     - --

     ((  ((((((                    ))))))   ))))))
161  GAAGAGGCUGCCGCUCUCGUCUGAAGGAACCAGCCUGGCUCAGCGUGCUGCUGUGCAGCGGCCAGGCUUCAUCUCUAACA
161  -AA-A---------U-UAG-C--AAU-A-----------------U--U-------AG-----A------A-----AAC-
     -  - --------- -   - --   - ----------------- -- -------  ----- ------ -----   -


241  GCAGCGAACGGACAUGAGGGAGCGCAAACCCUCGUCCCAAACAUCAUGAAUGCGUCGCACGGGCUCCAGCGUCAGGGGCC
241  -CAGCU--C-----U-AGU-AG---------UC-U---AG-C--C-UGA-U---UC--A-GU--U--AUU-UC-------
     -     -- ----- -   -  ---------  - ---  - -- -   - ---  -- -  -- --   -  -------

                 ((((((((   ((( [[[[[[[)))))))  ))))          ]]]]]]]   )))))    -
321  AGAGGUGGGAAACGGUGUGAAACUCCUGUCCUGGGGAUCACCGACCGAUACGCCAAACCAGGACUACGCCCGUAG-AACG
321  -----U---AAA---U-U--A--U--U-U-------AU----G----UUAUGUUAUU-------UAAGCUUGUAGUAACU
     ----- ---   --- - -- -- -- - -------  ---- ----          -------   )))))       )

     --)-))) -        ))))-)   ((-(((       )))-)))))))))
401  --C-AGUG-AGAAAGACACCA-GGACAG-GGGUUCGAGUCCC-CUCAGGUCCACgu
401  AUCUAGUGUACAAUUUC-U-AUGGAC-GUGGGUUCAAUUCCCAC-CAGCUCCACaa
     ))))))))))))))) -)-)))   -(((((       )))))-)))))))
```

Figure 5.14: Constrained alignment between MB and CC

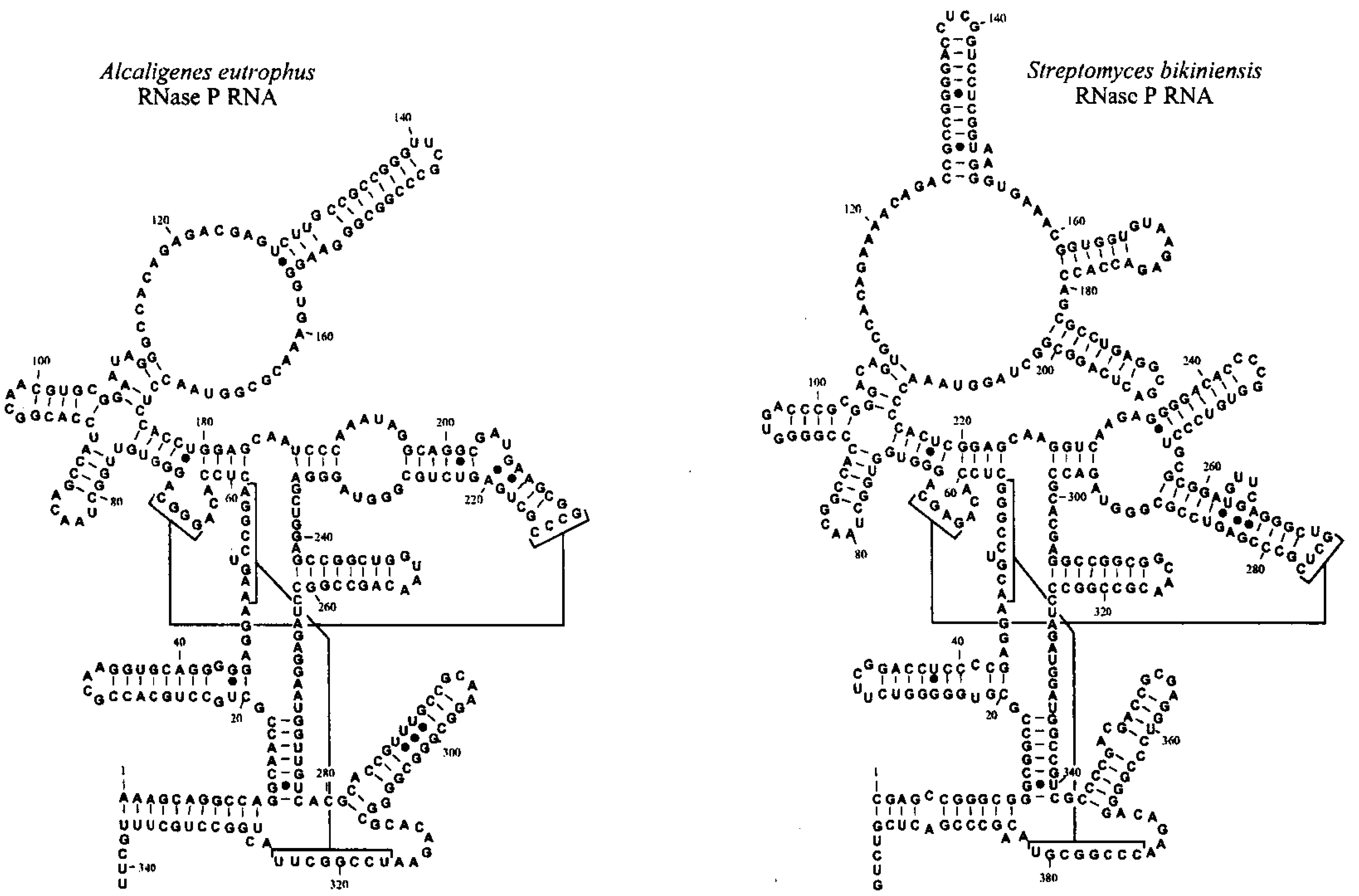


Figure 5.15: Structures of *Alcaligenes eutrophus* and *Streptomyces bikiniensis* from the RNase P database. Source: http://www.mbio.ncsu.edu/RNaseP

The RNA tertiary structures that we used above are very simple. Now we perform experiments on two moderate RNA tertiary structures. These RNA structures are selected from the RNase P Database [2]. Ribonuclease P is the ribonucleoprotein endonuclease that cleaves transfer RNA precursors, removing $5'$ precursor sequences and generating the mature $5'$ terminus of the tRNA. *Alcaligenes eutrophus* (AE) is from the beta purple bacteria group and *Streptomyces bikiniensis* (SB) is from the high G+C gram positive group. Figure 5.15 shows a 2D drawing of these two RNA structures. The secondary bondings are represented by a dash or a dot between two bases and tertiary bondings are represented by a solid line between distant bases. We have shown their region table representations in Figure 5.1.

The results of the second comparison are shown in Table 5.6 (n = sequence length, s = maximal number of arcs in crossing stem, pk = number of pseudoknots, fixed parameter k = number of crossing maximal stem matches that overlap in a common point, (i,j) = the start bases's positions of the two selected maximal stems in our constrained alignment algorithm).

					Wang and Zhang's results		Our results		
Aligned RNAs	n	s	k	pk	runtime	memory	(i,j)	runtime	memory
AE / SB	341/398	8/8	8	3/4	8s	2,972 KB	(12, 12)	34s	15,916KB
							(50, 48)	32s	16,028KB

Table 5.6: Second comparison of our results to Wang and Zhang's constrained alignment results

The concrete alignments between AE and SB produced by Wang and Zhang's algorithm and our algorithm are shown in Figure 5.16 and Figure 5.17, respectively. These alignments are exactly the same.

```
--------------------------------------------------------------------------------------
by Wang and Zhang's algorithm:

Alcaligenes-eutrophus-pb-b
Streptomyces-bikiniensis-gpb-h

score = 186.000000

     (((((((((((((((((( (( ((((((((    )))))))) ))    [[[ [[[[[((((   [[[[ ((((( ((((
  1  AAAGCAGGCCAGGCAACCGCUGCCUGCACCGCAAGGUGCAGGGGGAGGAAAGUCCGGACUCCACAGGCAGGGUGUUGGC
  1  CGAGCCGGGCGGGCGGCCGCGUGGGGGUCUUC--GGACCUCCCCGAGGAACGUCCGGGCUCCACAGAGCAGGGUGGUGGC
     (((((((((((((((((( (( (((((((   -- ))))))) ))    [[[ [[[[[((((   [[[[ ((((((((((

          )))) (((((    )))) )((   ((          -    (((( ((((((-((--    --))-))))--))
 81  UAACAGCCAUCCACGGCAACGUGCGGAAUAGGGCCACAGAGA-CGAGUCUUGCCGCCG-GG--UUCG--CC-CGGC--GG
 81  UAACGGCCACCCGGGGUGACCCGCGGGACAGUGCCACAGAAAAC-AG----ACCGCCGGGGACCUCGGUCCUCGGUAAGG
          )))))(((((    )))) )((   ((            -  ---- ((((((((((     ))))))))   ))

      --))))------  --  - ----- ----------  ---------- --     ))))))))))))    ((((  -
161  G--AAGG------GU--GA-A-----A----------CG----------C--GGUAACCUCCACCUGGAGCAAUCCCAA-
161  GUGAAACGGUGGUGUAAGAGACCACCAGCGCCUGAGGCGACUCAGGCGGCUAGGUAAACCCCACUCGGAGCAAGGUCAAG
            ((((((       ))))))  ((((((((    ))))))))         ))))))))))))    ((((

       --- ------ ----------((((((   -((((-( ]]]])-))))))))))     ))))      -((((((((
241  AU---A------G----------GCAGGCGAU-GAAG-CGGCCCG-CUGAGUCUGCGGGUAGGAGCUGGA-GCCGGCUG
241  AGGGGACACCCCGGUGUCCCUGCGCGGAUGUUCGAGGGCUGCUCGCCCGAGUCCGCGGGUAGACCGCACGAGGCCGGC-G
      ((((((((    ))))))))  ((((((    (((((( ]]]]))))))))))))     ))))      (((((((-(

         ))))))))-     -     )))))))  (( (((((((-((     ))-))))))) ))        ]]]]]]]]
321  GUAACAGCCGGC-CUAGA-GGAAUGGUUGUCACGCACCGUUUG-CCGCAAGG-CGGGCGGGGCGCACAGAAUCCGGCUUA
321  GCAAC-GCCGGCCCUAGAUGGA-UGGCCGUCGC-C-CCGAC-GACCGCGAGGUCC--CGG-G-G-ACAGAACCCGGCGUA
         )-)))))))         - ))))))) (-(-(((  -((((    )))) --)))-)-)-      ]]]]]]]]

     ) ))))))))))
401  UCGGCCUGCUUUGCUU
401  CAGCCCGACUCGUCUG
     ) ))))))))))
```

Figure 5.16: Constrained alignment between AE and SB produced by Wang and Zhang's algorithm

```
------------------------------------------------------------------------------------------
by our algorithm:

sequence 1: Alcaligenes-eutrophus-pb-b
sequence 2: Streptomyces-bikiniensis-gpb-h
The constrained stem pair:
1st stem:(outermost arc, innermost arc), 2nd stem:(outermost arc, innermost arc)
          (( 50, 324), ( 58, 317)),             (( 48, 381), ( 56, 374))

The optimal constrained alignment score is 186.000
The optimal constrained alignment is:

      (((((((((((((((((( (( ((((((((    )))))))) ))    [[[ [[[[[((((   [[[[ ((((( ((((
   1  AAAGCAGGCCAGGCAACCGCUGCCUGCACCGCAAGGUGCAGGGGGAGGAAAGUCCGGACUCCACAGGGCAGGGUGUUGGC
   1  CGAGCCGGGCGGGCGGCCGCGUGGGGGUCUUC--GGACCUCCCCGAGGAACGUCCGGGCUCCACAGAGCAGGGUGGUGGC
      (((((((((((((((((( (( (((((((   -- ))))))) ))    [[[ [[[[[((((   [[[[ ((((((((((

           )))) (((((    )))) )((   ((          -    (((( ((((((-((--    --))-))))--))
  81  UAACAGCCAUCCACGGCAACGUGCGGAAUAGGGCCACAGAGA-CGAGUCUUGCCGCCG-GG--UUCG--CC-CGGC--GG
  81  UAACGGCCACCCGGGGUGACCCGCGGGACAGUGCCACAGAAAAC-AG----ACCGCCGGGGACCUCGGUCCUCGGUAAGG
           )))))(((((    )))) )((   ((            -  ---- (((((((((((    )))))))))  ))

       --))))------  --  - ----- ----------  ---------- --     )))))))))))))   ((((  -
 161  G--AAGG------GU--GA-A-----A----------CG----------C--GGUAACCUCCACCUGGAGCAAUCCCAA-
 161  GUGAAACGGUGGUGUAAGAGACCACCAGCGCCUGAGGCGACUCAGGCGGCUAGGUAAACCCCACUCGGAGCAAGGUCAAG
             ((((((       ))))))  ((((((((    ))))))))         )))))))))))))   ((((

        --- ------ ----------((((((   -((-((( ]]]])))-))))))))     ))))      -((((((((
 241  AU---A------G----------GCAGGCGAU-GA-AGCGGCCCGCU-GAGUCUGCGGGUAGGGAGCUGGA-GCCGGCUG
 241  AGGGGACACCCCGGUGUCCCUGCGCGGAUGUUCGAGGGCUGCUCGCCCGAGUCCGCGGGUAGACCGCACGAGGCCGGC-G
       ((((((((    ))))))))  ((((((    (((((( ]]]]))))))))))))     ))))      (((((((-(

          ))))))))-     -     )))))))  (( (((((((-((    ))-))))))) ))         ]]]]]]]
 321  GUAACAGCCGGC-CUAGA-GGAAUGGUUGUCACGCACCGUUUG-CCGCAAGG-CGGGCGGGGCGCACAGAAUCCGGCUUA
 321  GCAAC-GCCGGCCCUAGAUGGA-UGGCCGUCGC-C-CCGAC-GACCGCGAGGUCC--CGG-G-G-ACAGAACCCGGCGUA
          )-)))))))         - ))))))) (-(-(((  -((((    )))) --)))-)-)-       ]]]]]]]

      ) ))))))))))
 401  UCGGCCUGCUUUGCUU
 401  CAGCCCGACUCGUCUG
      ) ))))))))))

------------------------------------------------------------------------------------------
by our algorithm:

sequence 1: Alcaligenes-eutrophus-pb-b
sequence 2: Streptomyces-bikiniensis-gpb-h
The constrained stem pair:
1st stem:(outermost arc, innermost arc), 2nd stem:(outermost arc, innermost arc)
          (( 12, 278), ( 18, 272)),             (( 12, 341), ( 18, 335))

The optimal constrained alignment score is 186.000
The optimal constrained alignment is:

      (((((((((((((((((( (( ((((((((    )))))))) ))    [[[ [[[[[((((   [[[[ ((((( ((((
   1  AAAGCAGGCCAGGCAACCGCUGCCUGCACCGCAAGGUGCAGGGGGAGGAAAGUCCGGACUCCACAGGGCAGGGUGUUGGC
   1  CGAGCCGGGCGGGCGGCCGCGUGGGGGUCUUC--GGACCUCCCCGAGGAACGUCCGGGCUCCACAGAGCAGGGUGGUGGC
      (((((((((((((((((( (( (((((((   -- ))))))) ))    [[[ [[[[[((((   [[[[ ((((((((((

           )))) (((((    )))) )((   ((          -    (((( ((((((-((--    --))-))))--))
  81  UAACAGCCAUCCACGGCAACGUGCGGAAUAGGGCCACAGAGA-CGAGUCUUGCCGCCG-GG--UUCG--CC-CGGC--GG
  81  UAACGGCCACCCGGGGUGACCCGCGGGACAGUGCCACAGAAAAC-AG----ACCGCCGGGGACCUCGGUCCUCGGUAAGG
           )))))(((((    )))) )((   ((            -  ---- (((((((((((    )))))))))  ))

       --))))------  --  - ----- ----------  ---------- --     )))))))))))))   ((((  -
 161  G--AAGG------GU--GA-A-----A----------CG----------C--GGUAACCUCCACCUGGAGCAAUCCCAA-
 161  GUGAAACGGUGGUGUAAGAGACCACCAGCGCCUGAGGCGACUCAGGCGGCUAGGUAAACCCCACUCGGAGCAAGGUCAAG
             ((((((       ))))))  ((((((((    ))))))))         )))))))))))))   ((((

        --- ------ ----------((((((   -((-((( ]]]])))-))))))))     ))))      -((((((((
 241  AU---A------G----------GCAGGCGAU-GA-AGCGGCCCGCU-GAGUCUGCGGGUAGGGAGCUGGA-GCCGGCUG
 241  AGGGGACACCCCGGUGUCCCUGCGCGGAUGUUCGAGGGCUGCUCGCCCGAGUCCGCGGGUAGACCGCACGAGGCCGGC-G
       ((((((((    ))))))))  ((((((    (((((( ]]]]))))))))))))     ))))      (((((((-(

          ))))))))-     -     )))))))  (( (((((((-((    ))-))))))) ))         ]]]]]]]
 321  GUAACAGCCGGC-CUAGA-GGAAUGGUUGUCACGCACCGUUUG-CCGCAAGG-CGGGCGGGGCGCACAGAAUCCGGCUUA
 321  GCAAC-GCCGGCCCUAGAUGGA-UGGCCGUCGC-C-CCGAC-GACCGCGAGGUCC--CGG-G-G-ACAGAACCCGGCGUA
          )-)))))))         - ))))))) (-(-(((  -((((    )))) --)))-)-)-       ]]]]]]]

      ) ))))))))))
 401  UCGGCCUGCUUUGCUU
 401  CAGCCCGACUCGUCUG
      ) ))))))))))
```

Figure 5.17: Constrained alignment between AE and SB produced by our algorithm

Chapter 6

Conclusions and Future Work

In this thesis, based on the previous works on the alignment between RNA structures, we have presented an improved algorithm for alignment between RNA tertiary structures. For simple tertiary structures, we can compute the optimal alignment efficiently. For moderate tertiary structures, we adopt the constrained alignment approach. Although the result produced by constrained alignment is not guaranteed to be an optimal solution, in practice it would be reasonable.

Major contributions of this thesis are summarized as follows:

- We proposed a new partition approach of the set of maximal stem pairs. And we proposed a method to optimize the preliminary partition result. The optimized partition is local minimal.

- We proposed a method to preprocess "crossing stem pairs" and filter out unnecessary "crossing stem pairs" to accelerate the computation of optimal alignment between RNA tertiary structures.

- We proposed a faster implementation to compute optimal alignment between RNA tertiary structures.

- We proposed a constrained alignment method to align moderate RNA tertiary structures.

These algorithms have been implemented into a software package. We performed extensive experiments of our alignment algorithms on real RNA structures. Experimental tests show that our algorithms can be used to compute alignment between RNA tertiary structures in practical applications.

For further work, we have several directions listed as follows.

- We can get local minimal partition of the set of maximal stem pairs by our method. But currently we have not found any polynomial algorithm to compute the minimum partition of the set of maximal stem pairs. We conjecture that this problem is NP-Hard.
- Currently, our alignment algorithm is under linear gap penalty model. It can be extended to affine gap penalty model.
- In the current implementation, we only accept the region table format for input RNA tertiary structures. We can extend our package to accept other RNA data formats (ct2 format, rnaml format).
- We can implement the optimization technique discussed in Section 4.5.10 to remove redundant computation.
- We can design artificial RNA tertiary structures to test how complicated tertiary structures that our optimal alignment algorithm can handle with limited memory (such as 1 GB). We can set a standard according to those designed tertiary structures. For tertiary structures which are simpler than the designed structures, we can compute the optimal alignment. For tertiary structures which are more complicated than the designed structures, we can compute the constrained alignment.

These issues are worth further investigation.

Bibliography

[1] V. Bafna, S. Muthukrishnan, and R. Ravi, "Computing similarity between RNA strings", *Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching (CPM 1995)*, LNCS 937, pp.1-14, 1995.

[2] J. W. Brown, "The Ribonuclease P Database", *Nucleic Acids Research*, 27:314, 1999.

[3] J. H. Chen, S. Y. Le, and J. V. Maizel, "Prediction of common secondary structures of RNAs: a genetic algorithm approach", *Nucleic Acids Research*, 28:4, pp.991-999, 2000.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms* (2nd edition), The MIT Press, 2001.

[5] F. Corpet and B. Michot, "RNAlign program: alignment of RNA sequences using both primary and secondary structures", *Computer Applications in the Biosciences*, vol. 10, no. 4, pp.389-399, 1995.

[6] P. A. Evans, "Algorithms and Complexity for Annotated Sequence Analysis", *PhD thesis, University of Victoria*, 1999.

[7] O. Gotoh, "An Improved Algorithm for Matching Biological Sequences", *Journal of Molecular Biology*, 162, pp.705-708, 1982.

[8] T. Jiang, G. Lin, B. Ma, and K. Zhang, "The longest common subsequence problem for arc-annotated sequences", *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching (CPM 2000)*, LNCS 1848, pages 154-165, 2000.

[9] T. Jiang, G. Lin, B. Ma, and K. Zhang, "A general edit distance between two RNA structures", *Journal of Computational Biology*, 9(2): 371-388, 2002.

[10] H. T. Laquer, "Asymptotic limits for a two-dimensional recursion", *Studies in Applied Mathematics*, 64: 271-277, 1981.

[11] H. Lenhof, K. Reinert, and M. Vingron, "A polyhedral approach to RNA sequence structure alignment", *Proceedings of the Second Annual International*

Conference on Computational Molecular Biology (RECOMB'98), pages 153-159, 1998.

[12] B. Ma, L. Wang, K. Zhang, "Computing similarity between RNA structures", *Theoretical Computer Science*, 276(1-2): 111-132, 2002.

[13] M. Möhl, S. Will, and R. Backofen, "Fixed parameter tractable alignment of RNA structures including arbitrary pseudoknots", *Proceedings of the 19th Annual Symposium on Combinatorial Pattern Matching (CPM 2008)*, LNCS 5029, pp. 69-81, 2008.

[14] S. E. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino-acid sequences of two proteins", *Journal of Molecular Biology*, vol. 48, pp.443-453, 1970.

[15] C. H. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes", *Journal of Computer and System Sciences*, vol. 43, pp.425-440, 1991.

[16] Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjölander, R. Underwood, and D. Haussler, "Stochastic context-free grammar for tRNA modeling", *Nucleic Acids Research*, vol. 22, no. 23, pp. 5112-5120, 1994.

[17] D. Sankoff, "Simultaneous solution of the RNA folding, alignment and protosequence problems", *SIAM Journal on Applied Mathematics*, vol. 45, no. 5, pp.810-824, 1985.

[18] T. F. Smith and M. S. Waterman, "Comparison of Biosequences", *Advances in Applied Mathematics*, 2, pp.482-489, 1981.

[19] Z. Wang and K. Zhang, "Alignment between two RNA structures", *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001)*, LNCS 2136, pp. 690-703, 2001.

[20] K. Zhang, "Sequence Similarity", *Course Notes on Computational Biology*, University of Western Ontario, 2008.

[21] K. Zhang, L. Wang, and B. Ma, "Computing similarity between RNA structures", *Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching (CPM 1999)*, LNCS 1645, pp. 281-293, 1999.

[22] C. Zwieb, J. Gorodkin, B. Knudsen, J. Burks, J. Wower, "tmRDB (tmRNA database)", *Nucleic Acids Research*, 31(1), 446-447, 2003.