A General Edit Distance between RNA Structures *

Tao Jiang † Guohui Lin ‡ Bin Ma § Kaizhong Zhang ¶

June 26, 2001

Abstract

Arc-annotated sequences 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. Since RNA functions are largely determined by molecular confirmation and therefore secondary and tertiary structures, the comparison between RNA secondary and tertiary structures has received much attention recently. In this paper, we propose the notion of edit distance to measure the similarity between two RNA secondary and tertiary structures, by incorporating various edit operations performed on both bases and arcs (*i.e.* base-pairs). Several algorithms are presented to compute the edit distance between two RNA sequences with various arc structures and under various score schemes, either exactly or approximately with provably good performance. Preliminary experimental tests confirm that our definition of edit distance and the computation model are among the most reasonable ones ever studied in the literature.

Keywords: sequence annotation, string edit, dynamic programming, approximation algorithm, time efficiency.

---

*A preliminary version entitled "Edit distance between two RNA structures", by G.-H. Lin, B. Ma, and K. Zhang, appears in *Proceedings of the Fifth Annual International Conference on Computational Molecular Biology (RECOMB'01)*, pages 211–220, 2001.

†Supported in part by a UCR startup grant and NSF Grants CCR-9988353 and ITR-0085910. Department of Computer Science, University of California, Riverside, CA 92521. Email: jiang@cs.ucr.edu. On leave from McMaster University.

‡Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E1, Canada. Work done while the author was with Department of Computer Science, University of Waterloo, and Department of Computing and Software, McMaster University, supported in part by NSERC Research Grant OGP0046613 and a CITO grant, and Bioinformatics Solutions Inc.. Email: ghlin@math.uwaterloo.ca.

§Supported in part by NSERC Research Grant OGP0046506 and a CGAT grant. Department of Computer Science, University of Western Ontario, London, Ontario N6A 5B7, Canada. E-mail: bma@csd.uwo.ca.

¶Supported in part by NSERC Research Grant OGP0046373. Department of Computer Science, University of Western Ontario, London, Ontario N6A 5B7, Canada. E-mail: kzhang@csd.uwo.ca.

1 Introduction

It has been realized that RNA is an important molecule which performs a wide range of functions in biological system. Among the most significant factors in the molecular mechanism involved in these functions, are the secondary and tertiary structural features of an RNA. Specifically, to a preserved function there corresponds a preserved molecular confirmation and therefore preserved secondary and tertiary structures. The comparison of RNA structures has received much attention recently in the literature. There are several ways to represent RNA structures and formulate corresponding similarity measures. One of them is to represent RNA secondary structures as (labeled or unlabeled) trees. There are several algorithms for computing the distance between two trees [19, 21, 24, 20, 25, 23, 22]. Another method for modeling RNA structures is using *stochastic context-free grammars* (SCFGs), which is a generalization of *hidden Markov models* (HMMs) capturing well the common primary and secondary structures [16, 4]. Notice that the above two methods do not treat base-pairs as units of comparison. Instead, they focus on local structures and treat them as subunits. Other work that does not treat base-pairs as units includes [5, 2, 12]. On the other hand, there are several papers which consider base-pairs as units, and discuss the similarity at a sequence level [17, 28, 27, 6, 8, 10, 14]. In this paper, we follow this last line of research and define a notion of edit distance between two RNA sequences, taking into consideration the primary, secondary, and tertiary structures.

An RNA *primary* sequence is a string $S \in \{\mathsf{A}, \mathsf{U}, \mathsf{C}, \mathsf{G}\}^*$. Supposing that $|S| = n$, the secondary and tertiary structures of S can be represented as a set P of *arcs*, where each arc is of form (i_1, i_2) for some $1 \leq i_1 < i_2 \leq n$ meaning that $S[i_1]$ (*i.e.* the i_1th base/letter in S) and $S[i_2]$ are two complementary bases connected by a bond. Bases $S[i_1]$ and $S[i_2]$ are said to form a *base-pair*. It is observed that a base could participate in *at most* one base-pair. Furthermore, the set of arcs representing a secondary structure is assumed to be *nested*, *i.e.* for any two arcs (i_1, i_2) and (i_3, i_4), either $i_1 < i_3 < i_4 < i_2$ (called *nested*) or $i_1 < i_2 < i_3 < i_4$ (called *independent*); the set of arcs representing a tertiary structure could be *crossing*, *i.e.* for any two arcs (i_1, i_2) and (i_3, i_4), it is possible that $i_1 < i_3 < i_2 < i_4$ (called *crossing*). Generally, a sequence S with a set P of arcs representing its structural information is called an *arc-annotated sequence*, denoted as (S, P). When the set of arcs is crossing/nested, we say that the arc annotated structure is crossing/nested and the sequence is a crossing/nested arc-annotated sequence. In the most special case when there are no arcs at all, *i.e.* when a primary sequence is considered, the sequence is called a *plain* (arc-annotated) sequence. In other words, we use crossing (nested, or plain) arc-annotated sequences over the alphabet $\{\mathsf{A}, \mathsf{U}, \mathsf{C}, \mathsf{G}\}$ to represent the RNA tertiary (secondary, or primary, respectively) structures.

Following the notion of similarity between two plain sequences, we define the similarity between two arc-annotated sequences below. One common way to measure the similarity between two plain sequences [9] is using the *edit distance*, which focuses on the cost of transforming/editing one sequence into the other. The permitted edit operations are *insertion* of a base into the first sequence, *deletion* of a base from the first sequence, and *substitution* (or *replacement*) of a base in the first sequence with a base in the second sequence. A deletion and an insertion have the same cost w_d, and a substitution has cost w_m. The edit distance between two plain sequences is the minimum cost of the edit transcripts, each of which describes a series of edit operations that transform one sequence into the other. The cost of an edit transcript is defined to be the sum of costs of the edit operations therein.

The edit distance between two plain sequences, along with an optimal edit transcript, can be computed in $O(nm)$ [13, 18] time, where n and m are the lengths of the two sequences. To formalize the edit distance between two arc-annotated sequences, we need to specify the types of edit operations that can be performed on arcs and bases.

1.1 Edit Operations and Problem Description

For two plain sequences S_1 and S_2, an edit transcript also specifies an alignment of S_1 and S_2 in a straightforward way. Conversely, given a sequence alignment, it is easy to write down the corresponding edit transcript. For the ease of exposition, we define the edit distance between two arc-annotated sequences via a sequence alignment instead of an edit transcript. Notice that a sequence alignment of arc-annotated sequences specifies how the arcs are aligned and thus it is also a structural alignment.

In the following, consider two arc-annotated sequences (S_1, P_1) and (S_2, P_2) over some fixed alphabet Σ and a specific sequence alignment M. A base $S_1[i]$ is said *free* if there is no arc incident on it. Let us first look at the edit operations performed on arcs and its incident bases as specified by M. Suppose that $(i_1, i_2) \in P_1$ and $(j_1, j_2) \in P_2$ are arcs such that $S_1[i_1]$ is aligned with $S_2[j_1]$ and $S_1[i_2]$ is aligned with $S_2[j_2]$. If $S_1[i_1] = S_2[j_1]$ and $S_1[i_2] = S_2[j_2]$, then they form an *arc-match*; if at least one of the equalities does not hold, then they form an *arc-mismatch*. If $(i_1, i_2) \in P_1$ is an arc such that $S_1[i_1]$ and $S_1[i_2]$ are both aligned with some bases in S_2, say $S_2[j_1]$ and $S_2[j_2]$, but $(j_1, j_2) \notin P_2$, then there is an *arc-breaking* operation which breaks arc (i_1, i_2) and leaves $S_1[i_1]$ and $S_1[i_2]$ free. Biologically, this may happen when mutations nearby cause the force change and ultimately make the bond between $S_1[i_1]$ and $S_1[i_2]$ too weak to exist anymore. If $(i_1, i_2) \in P_1$ is an arc such that one of $S_1[i_1]$ and $S_1[i_2]$, say $S_1[i_1]$, is aligned with some base in S_2, say $S_2[j_1]$, while the other (that is, $S_1[i_2]$) is aligned with a space (*i.e.* nothing), then there is an *arc-altering* operation in which base $S_1[i_2]$ is deleted and thus arc (i_1, i_2) is broken. Such an operation leaves base $S_1[i_1]$ free. The operation models the scenario where one of the bases in a base-pair disappears during the evolution. If $(i_1, i_2) \in P_1$ is an arc such that both $S_1[i_1]$ and $S_1[i_2]$ are aligned with spaces, then there is an *arc-removing* operation which breaks arc (i_1, i_2) and deletes both the bases. This operation models the scenario both bases in a base-pair disappear during the evolution. The above definitions apply to the arcs in (S_2, P_2) as well.

After considering the above edit operations on arcs and their incident bases as specified by the alignment M, we discuss base operations that have not been covered by these arc operations. If base $S_1[i]$ is aligned with base $S_2[j]$ for some j and $S_1[i] = S_2[j]$, then we call $\langle S_1[i], S_2[j] \rangle$ a *base-match* if it is not involved in any arc operation. If, however, $S_1[i] \neq S_2[j]$, then $\langle S_1[i], S_2[j] \rangle$ is called a *base-mismatch*, which indicates a single base mutation. If a base is aligned with a space, then it corresponds to a *base-deletion* (or, *base-insertion*). Note that the bases in these (base) operations are not necessarily free. For example, an arc-breaking or arc-altering operation may be followed by a (base) operation involving some base who is incident on the arc.

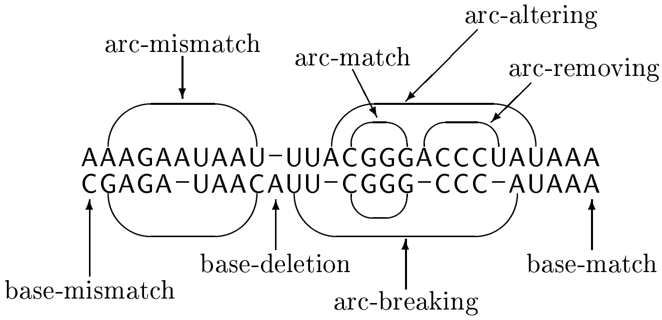

Figure 1: A sequence alignment and the allowed edit operations.

A base-match and an arc-match cost nothing. A base-mismatch has cost w_m, and an arc-mismatch has cost $\frac{w_{am}}{2}$ or w_{am} depending on the two arcs involved. Recall that when an arc $(i_1, i_2) \in P_1$ is aligned with an arc $(j_1, j_2) \in P_2$, then $S_1[i_1]$ is aligned with $S_2[j_1]$ and $S_1[i_2]$ is aligned with $S_2[j_2]$. If $S_1[i_1] \neq S_2[j_1]$ or $S_1[i_2] \neq S_2[j_2]$, then they form an *arc-mismatch*; If exactly one of the above is equality, then the cost is $\frac{w_{am}}{2}$ whereas if $S_1[i_1] \neq S_2[j_1]$ and $S_1[i_2] \neq S_2[j_2]$, then the cost is w_{am}. A base-deletion has cost w_d, an arc-breaking has cost w_b, an arc-altering has cost w_a, and an arc-removing has cost w_r which is usually assumed to be at least as large as w_d (*i.e.* $w_r \geq w_d$). The above 6 operations form the legal edit operations (see Figure 1 for a simple illustration). Notice that it is straightforward to determine an optimal series of edit operations corresponding to any given sequence alignment. The key idea is to determine the operations performed on arcs first and then those on single bases. For example, suppose that in the alignment M, there exist arcs $(i_1, i_2) \in P_1$ and $(j_1, j_2) \in P_2$, $S_1[i_1]$ and $S_2[j_1]$ are each aligned with a space, and $S_1[i_2]$ is aligned with $S_2[j_2]$ but $S_1[i_2] \neq S_2[j_2]$. Then there are two arc-altering operations and a base-mismatch associated with them and these operations have a total cost of $2w_a + w_m$. The sum of the costs of edit operations specified by an alignment is the cost associated with the alignment. The edit distance between two arc-annotated sequences is defined to be the minimum cost of the alignments of the two sequences. Under a fixed score scheme specifying parameters $w_m, w_{am}, w_d, w_b, w_a, w_r$, the EDIT DISTANCE PROBLEM FOR ARC-ANNOTATED SEQUENCES is, given a pair of arc-annotated sequences, to compute the edit distance between them, along with an optimal sequence alignment (or equivalently an optimal series of edit operations that transform the first sequence into the second one).

Since the difficulty of the problem depends on the complexity of the arc structures of the input sequences, apart from the score scheme, we use EDIT(TYPE$_1$, TYPE$_2$) to denote the EDIT DISTANCE PROBLEM FOR ARC-ANNOTATED SEQUENCES where the arc structure of the first input sequence is of TYPE$_1$ and the arc structure of the second input sequence is of TYPE$_2$. Notice that there are three possible types of arc structure: crossing, nested, and plain, from high complexity to low. We assume without loss of generality that TYPE_1 is always higher than or equal to TYPE_2. This way, we have 6 distinct edit distance problems, from the most complicated to the least:

- EDIT(CROSSING, CROSSING),
- EDIT(CROSSING, NESTED),
- EDIT(CROSSING, PLAIN),
- EDIT(NESTED, NESTED),
- EDIT(NESTED, PLAIN), and
- EDIT(PLAIN, PLAIN).

EDIT(PLAIN, PLAIN) is well-known and has been completely solved [13, 18].

1.2 Related Work

Zhang *et al.*, in a series of papers, have considered a similarity measure between two RNA secondary structures that is very close to ours. In a recent work [27], the authors studied a restricted problem where if an arc of P_1 is not matched or mismatched with some arc in P_2, then this arc has to be removed. In other words, the edit operations arc-breaking and arc-altering are not allowed in the model. Under this restricted assumption, they were able to show that computing the edit distance between two RNA tertiary structures (*i.e.* EDIT(CROSSING,CROSSING)) is NP-hard, and the other 5 cases are all solvable in polynomial time. However, as we shall also see in Section 5, in their model an arc-breaking operation has to be formulated as an arc-removing operation plus two base-deletions and an arc-altering operation has to be formulated as an arc-removing operation plus a base-deletion. It is sometimes more reasonable to assume that because of mutations in the neighboring segments (or even more distant segments in some cases), the bond between a base-pair may become loose and finally broken. In this scenario, an arc-breaking (or arc-altering) operation is much more plausible than first deleting a base-pair and then inserting two single bases (or one single base, respectively).

Another line of work related to similarity comparison between RNA secondary and tertiary structures is focussed on primary sequences [2, 5] where the comparison is basically done on the primary sequences 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. Our model treats base-pairs as basic units and is closer to the general spirit of the comparative analysis methods currently being used in the analysis of RNA secondary and tertiary structures either manually or automatically.

1.3 Our Contributions and Roadmap

We consider the first five edit distance problems involving at least one RNA secondary or tertiary structure. We will present a dynamic programming algorithm for the problem EDIT(NESTED, PLAIN). Letting $|S_1| = n$ and $|S_2| = m$, this algorithm runs in $O(nm^3)$ time. It turns out that problem EDIT(CROSSING, PLAIN) is already MAX SNP-hard (and we will prove it here). Therefore, it is unlikely that problem EDIT(CROSSING, PLAIN) could be approximated within arbitrary ratio. This hardness result also implies that problems EDIT(CROSSING, CROSSING) and EDIT(CROSSING, NESTED) are MAX SNP-hard. We establish these hardness results under a wide range of score schemes. Nonetheless, we will show that under a class of *reasonable* score schemes, problem EDIT(CROSSING, NESTED) is solvable in polynomial time. We will explicitly design an $O(mn^3)$ time algorithm for these reasonable score schemes by extending the dynamic programming algorithm for problem EDIT(NESTED, PLAIN), where $|S_1| = n$ and $|S_2| = m$. It also turns out that under a wide range of score schemes, this polynomial time algorithm forms a provably good approximation algorithm for problem EDIT(CROSSING, NESTED). Specifically, it is an approximation algorithm with worst-case performance ratio $\max\left\{\frac{2w_a}{w_b+w_r}, \frac{w_b+w_r}{2w_a}\right\}$. Moreover, we show how to improve the running time of this approximation algorithm to $O(mn^2)$. We then perform a preliminary experimental study of our algorithm on the comparison of RNA secondary structures, that is, on solving problem EDIT(NESTED, NESTED), under a reasonable score scheme. The study shows that the algorithm outputs more sensible alignments of RNA secondary structures than that obtained by the algorithms in [27]. Further experimental study is under way.

The rest of the paper is organized as follows: In Section 2, we present the hardness results. Section 3 is devoted to the $O(nm^3)$ time dynamic programming algorithm designed for problem EDIT(NESTED, PLAIN). The approximation algorithm for EDIT(CROSSING, NESTED), extending the exact dynamic programming algorithm, is described in Section 4, where we also show that it computes exactly the edit distance under a class of reasonable score schemes and how to improve the speed of the algorithm. Some preliminary experimental study results on three pairs of RNAs with known secondary structures are presented in Section 5. Section 6 closes the paper with several concluding remarks.

2 Hardness Results

Given a graph $G = (V, E)$, a partition (V_1, V_2) of vertex set V satisfies $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. A *cut* of G is a set of edges each of which has one endpoint in V_1 and the other endpoint in V_2, where (V_1, V_2) is a partition of V, and it is denoted by $E(V_1, V_2)$.

> MAX-CUT-3: Given a cubic graph $G = (V, E)$, *i.e.* every vertex has degree 3, find a partition of V associated with the maximum cardinality cut.

For a maximization (or minimization) problem Π, we say it admits an α-approximation algorithm $\mathcal{A}$ if $\mathcal{A}$ runs in polynomial time, and for every instance of Π, the solution output by $\mathcal{A}$ is greater than or equal to a factor $\frac{1}{\alpha}$ (less than or equal to a factor α, respectively) of the optimum. In this case, Π is also said to be approximable within ratio α. It is well known that unless P = NP, Π is not approximable within ratio $1 + \epsilon$ for some positive ϵ, or equivalently, Π does not admit a *polynomial time approximation scheme* (PTAS), if Π is MAX SNP-hard.

Lemma 2.1 MAX-CUT-3 *is approximable within ratio* 1.1383 [7], *but MAX SNP-hard* [15, 1].

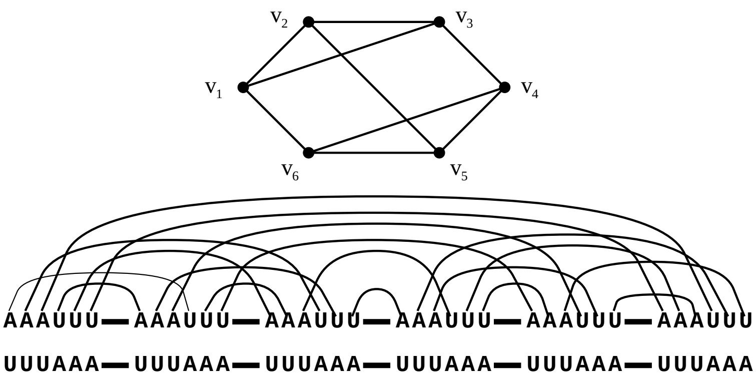

Figure 2: L-reduction: instance construction.

Theorem 2.2 EDIT(CROSSING, PLAIN) *is MAX SNP-hard.*

PROOF. We construct an L-reduction from MAX-CUT-3 in the following. The score schemes considered here satisfy that $w_m > w_d$ and $2w_a > w_b + w_r$. Notice that since the second sequence is plain, there is no arc-mismatch operation (and thus we wouldn't bother the value of w_{am}).

Given a cubic graph $G = (V, E)$, suppose that vertices in V are $v_0, v_1, \cdots, v_{n-1}$. For every vertex construct a segment of 6 bases in each of the two RNA primary sequences. Specifically, in S_1 the segment is "AAAUUU" and in S_2 the segment is "UUUAAA". In between two segments there is a segment of sufficiently many C's, which depends on the ratio $\frac{\min\{w_b, w_a, w_r\}}{w_d}$. Roughly speaking, these in-between segments prohibit the kind of optimal alignments in which some base in the segment for vertex v_i is aligned with some base in the segment for vertex v_j, while $i \neq j$. Notice that under a specified score scheme, this number could be determined in constant time and is polynomial in the length of input, and for ease of exposition we denote it by c. For example, we may take $c = \max\left\{2nw_d, \frac{\min\{w_b, w_a, w_r\}}{w_d}\right\}$. This way,

- $S_1 = (\mathsf{AAAUUU}(\mathsf{C})^c)^{n-1}\mathsf{AAAUUU}$, and
- $S_2 = (\mathsf{UUUAAA}(\mathsf{C})^c)^{n-1}\mathsf{UUUAAA}$.

For every edge $(v_i, v_j) \in E$ $(i < j)$, in P_1 there is an arc connecting one A in $S_1[(6+c)i+1, (6+c)i+3]$, the segment constructed corresponding to vertex v_i, and one U in $S_1[(6+c)j+4, (6+c)j+6]$, the segment constructed corresponding to vertex v_j, and an arc connecting one U in $S_1[(6+c)i+4, (6+c)i+6]$ and one A in $S_1[(6+c)j+1, (6+c)j+3]$. With three A's and three U's in every segment constructed corresponding to a vertex, it is trivial to make sure that at each base in sequence S_1, there is at most one incident arc. This gives us an instance of EDIT(CROSSING, PLAIN), where $|S_1| = |S_2| = 6n + c(n-1)$, $|P_1| = 3n$, and $|P_2| = 0$. Figure 2 illustrates the construction for a cubic graph containing 6 vertices, where a thick bar in between two segments stands for a segment of sufficiently many C's.

As we said before, by the abundance of base C's in the in-between segments, in any optimal alignment, the two in-between segments inserted between segments constructed for vertices v_i and v_{i+1} (where $0 \leq i \leq n-2$) in two sequences, respectively, must be aligned opposite to each other, or equivalently speaking, every base C is aligned with a base C. Further notice that if there is a base-mismatch between an A and a U (or a U and an A) from the segments constructed corresponding to vertex v_i, then it is always possible to replace it with a base-deletion and a base-match, while not changing any other edit operations. From the fact that $w_m > w_d$, we conclude there is no base-mismatch possible in the optimal alignment.

Now let us look at how segments $S_1[(6+c)i+1, (6+c)i+6]$ and $S_2[(6+c)i+1, (6+c)i+6]$ are aligned, without any base-mismatch. There are two types of possible local optimal alignments: In the A-type alignment, those three base A's are aligned with three base A's; In the U-type alignment, those three base U's are aligned with three base U's. If segments $S_1[(6+c)i+1, (6+c)i+6]$ and $S_2[(6+c)i+1, (6+c)i+6]$ are aligned in A-type (U-type, respectively), then we say that vertex v_i is in A-type (U-type, respectively). Notice that for each edge $(v_i, v_j) \in E$, there are 4 bases in S_1 and 4 bases in S_2 and 2 arcs related to it. If v_i and v_j are in a same type, then regarding to these 8 bases and 2 arcs, we have to perform two arc-altering operations and two base-deletions. The cost associated with them is $2w_a + 2w_d$.

If v_i and v_j are in different types, then we have to perform one arc-breaking operation, one arc-removing operation, and two base-deletions. The cost associated them is $w_b + w_r + 2w_d$.

Suppose in the optimal alignment there are k edges each of which has one endpoint in A-type and the other endpoint in U-type. Then the cost of the alignment would be

$$\begin{aligned} & (w_b + w_r + 2w_d) \cdot k + (2w_a + 2w_d) \cdot (\tfrac{3}{2}n - k) \\ = \; & 3n(w_a + w_d) - (2w_a - w_b - w_r)k. \end{aligned}$$

By including the vertices in A-type into V_1 and the vertices in U-type into V_2, these k edges form the cut set associated with the partition (V_1, V_2). Since $2w_a > w_b + w_r$ and the maximum cut in G is at least $\frac{3}{4}n$, the above is an L-reduction. Therefore, EDIT(CROSSING, PLAIN) is MAX SNP-hard. □

Remark 2.3 *The above proof can be suitably modified to fit for the score scheme where $w_m \leq w_d$ and $2w_a > w_b + w_r$. It is also easy to modify it to fit for the score scheme where $2w_a < w_b + w_r$. Nonetheless, when $2w_a = w_b + w_r$, the efficient algorithm provided in* Section 4.1 *implies that it also solves exactly* EDIT(NESTED, PLAIN).

Notice that EDIT(CROSSING, PLAIN) is a subproblem of problem EDIT(CROSSING, NESTED).

Corollary 2.4 EDIT(CROSSING, NESTED) *is MAX SNP-hard.*

3 A Dynamic Programming Algorithm for EDIT(NESTED, PLAIN)

In this section, we study the problem EDIT(NESTED, PLAIN) and present an efficient dynamic programming algorithm. Not only does the problem of comparing an RNA secondary structure to a primary RNA sequence is related to the RNA secondary structure inference or prediction problem [26], the techniques and ideas developed here will be useful in the algorithms for comparing RNA secondary and tertiary structures presented in the next section.

Let (S_1, P_1) and $(S_2, \emptyset)$ be an instance of EDIT(NESTED, PLAIN), where P_1 is nested. Let $n = |S_1|$ and $m = |S_2|$. We describe a dynamic programming algorithm that computes exactly the edit distance between (S_1, P_1) and $(S_2, \emptyset)$, along with an optimal sequence alignment, in $O(nm^3)$ time. We remark that the algorithm works under any score scheme (note again, there are no arc-mismatch operations).

For any arc $u \in P_1$, let u_l and u_r denote its *left endpoint* and *right endpoint* ($u_l < u_r$), respectively. We use $u(i)$ to denote the arc in P_1 incident on position (*i.e.* base) i of sequence S_1. If $u(i)$ does not exist (*i.e.* there is no arc in P_1 incident on base i), we call position (or base) i *free*. Suppose that i and i', where $1 \leq i \leq i' \leq n$, are a pair of positions of S_1 such that each of the arcs of P_1 either contains the interval $[i, i']$, or is contained in, or is disjoint from it. Let $P_1[i, i']$ denote the subset of arcs of P_1 that are contained in the interval $[i, i']$. Let $DP(i, i'; j, j')$, where $1 \leq j \leq j' \leq m$, denote the edit distance between the pair of partial sequences $(S_1[i, i'], P_1[i, i'])$ and $(S_2[j, j'], \emptyset)$. For any two bases $S_1[i]$ and $S_2[j]$, $\chi(i, j) = 1$ if $S_1[i] \neq S_2[j]$ (a base-mismatch), or 0 otherwise (a base-match). The following recurrence relation for $DP(i, i'; j, j')$ would suffice for the development of a dynamic programming algorithm. The recurrence relation is defined for disjoint cases concerning the relative values of the indices. Note that, $DP(i, i'; j, j')$ is not defined for all combinations of i, i', j, j'. These undefined values will not be required in the computation of the recurrence relation.

Case 1: For any $1 \leq i \leq n$ and $1 \leq j \leq j' \leq m$,

$$DP(i, i-1; j, j') = (|j' - j| + 1) w_d.$$

Case 2: For any $1 \leq i \leq i' \leq n$ and $1 \leq j \leq m$ such that (i) $i = 1$ or $(i-1, i'') \in P_1$ for some $i'' > i'$ and (ii) each of the arcs of P_1 either completely contains the interval $[i, i']$, or is contained in it, or is disjoint from it,

$$DP(i, i'; j, j-1) = (|P_1[i, i']|) w_r + (i' + 1 - i - 2|P_1[i, i']|) w_d.$$

Case 3: For any arc $(i,i') \in P_1$ and $1 \le j \le j' \le m$,

$$DP(i,i';j,j') = \min \begin{cases} DP(i+1,i'-1;j+1,j'-1) + w_b \\ \qquad + (\chi(i,j)+\chi(i',j'))w_m, & \textbf{if } j < j' \text{ (arc–breaking)}, \\ DP(i+1,i'-1;j,j'-1) + w_a + \chi(i',j')w_m & \text{(arc–altering)}, \\ DP(i+1,i'-1;j+1,j') + w_a + \chi(i,j)w_m & \text{(arc–altering)}, \\ DP(i+1,i'-1;j,j') + w_r & \text{(arc–removing)}, \\ DP(i,i';j,j'-1) + w_d & \text{(base–deletion)}, \\ DP(i,i';j+1,j') + w_d & \text{(base–deletion)}. \end{cases}$$

Case 4: For any $1 \le i \le i' \le n$ and $1 \le j \le j' \le m$ such that (i) $(i,i') \notin P_1$, (ii) $i = 1$ or $(i-1,i'') \in P_1$ for some $i'' > i'$, and (iii) each of the arcs of P_1 either completely contains the interval $[i,i']$, or is contained in it, or is disjoint from it, we distinguish two sub-cases:

If i' is free, then

$$DP(i,i';j,j') = \min \begin{cases} DP(i,i'-1;j,j'-1) + \chi(i',j')w_m, \\ DP(i,i'-1;j,j') + w_d, \\ DP(i,i';j,j'-1) + w_d. \end{cases}$$

If i' is not free, *i.e.* $i' = u(i')_r$ by condition (iii) above, then

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

One can easily design a dynamic programming algorithm to compute $DP(i,i';j,j')$ based on the above recurrence relation. We first initialize $DP(i,i';j,j')$ when $i-i' = 1$ or $j-j' = 1$ according to Cases 1 and 2. Then we consider all combinations of $i,i';j,j'$ that satisfy the conditions of Cases 3 or 4, in the ascending order of $i'-i$, and calculate $DP(i,i';j,j')$ using an appropriate recurrence formula. The entry $DP(1,n;1,m)$ records the edit distance between (S_1,P_1) and $(S_2,\emptyset)$.

The time complexity of the algorithm can be analyzed as follows. Observe that computing each above recurrence formula takes $O(1)$ time except for one of the formulae in Case 4 which takes $O(m)$ time in the worst case. Clearly, Case 1 requires an overall time of $O(nm^2)$. Since the number of arcs in P_1 is at most $n/2$, Case 3 requires a total computation time of $O(nm^2)$ as well. For Cases 2 and 4, we observe that the total number of intervals $[i,i']$ considered in these two cases is bounded by n. This is because that for any two distinct arcs (i_1-1,i_1'') and (i_2-1,i_2'') in P_1, the sets of index pairs of the forms (i_1,i_1') and (i_2,i_2'), where $i_1' < i_1''$ and $i_2' < i_2''$, respectively, satisfying the conditions of Cases 2 and 4 are disjoint. It follows that these two cases require the computation of a total number of $O(nm^2)$ $DP(i,i';j,j')$ entries, and hence $O(nm^3)$ time.

By using a standard back-tracing technique, we can find an optimal alignment of S_1 and S_2 from the table storing the $DP(i,i';j,j')$ entries in an additional $O(nm^2)$ time. Therefore the overall algorithm runs in $O(nm^3)$ time. The correctness of the algorithm follows directly from the fact that we consider the two endpoints of an arc simultaneously.

The above discussion proves the following theorem:

Theorem 3.1 EDIT(NESTED, PLAIN) *is solvable in time* $O(nm^3)$.

4 Algorithms for EDIT(CROSSING, NESTED)

We first consider a class of score schemes satisfying a certain (reasonable) condition and obtain an efficient dynamic programming algorithm. The algorithm is then adapted to give an approximation algorithm for EDIT(CROSSING, NESTED) under a general score scheme.

4.1 A Solvable Case

It is not hard to see that in EDIT(CROSSING, NESTED), if the score scheme satisfies the condition that $2w_a = w_b + w_r$, then we may treat every arc-altering operation as an arc-breaking operation plus a base-deletion of which the cost is $\frac{w_r - w_b}{2}$, and treat every arc-removing operation as an arc-breaking operation plus two base-deletions each of which has cost again $\frac{w_r - w_b}{2}$. Therefore, by associating to each base-deletion involving a base incident on an arc a cost of $\frac{w_r - w_b}{2}$ (note that this value might be negative), we may discard the arc-altering and arc-removing operations while only keep the arc-breaking operation. In other words, only the arc-match operation, the arc-mismatch operation, and the arc-breaking operation are now required concerning arcs. Notice that each free base deletion still costs w_d. It turns out that we can design a dynamic algorithm similar to that for EDIT(NESTED, PLAIN) to compute the edit distance under this class of score schemes. The key is to distinguish between free bases and bases incident with arcs.

Let (S_1, P_1) and (S_2, P_2) be an instance of EDIT(CROSSING, NESTED), where P_2 is nested. Denote $n = |S_1|$ and $m = |S_2|$. For simplicity, let $\psi_1(i) = 1$ if there is an arc in P_1 incident on $S_1[i]$, and 0 otherwise. $\psi_2(j)$ is similarly defined for $S_2[j]$. Then, deleting base $S_1[i]$ costs $(1 - \psi_1(i))w_d + \psi_1(i)w'_d = w_d + \psi_1(i)(\frac{w_r - w_b}{2} - w_d)$, where $w'_d = \frac{w_r - w_b}{2}$ denotes the cost of deleting a base incident on some arc. The basic idea of the dynamic programming computation is very similar to that of the algorithm for EDIT(NESTED, PLAIN) described in the last section. However, the recurrence relation is quite different in details. In the following computation, when we know that some arc in P_1 has been broken, we will split the cost w_b equally between its two incident bases. That is, base $S_1[i]$ will be charged an additional cost of $\psi_1(i)\frac{w_b}{2}$, unless it is involved in an arc-match, or an arc-mismatch between a pair of the corresponding arcs. For example, deleting base $S_1[i]$ now costs $w_d + \psi_1(i)(\frac{w_r - w_b}{2} - w_d) + \psi_1(i)\frac{w_b}{2} = w_d + \psi_1(i)(\frac{w_r}{2} - w_d)$.

Case 1: For any $1 \le i \le i' \le n$ and $1 \le j \le m$,

$$DP(i, i'; j, j-1) = \sum_{h=i}^{i'} w_d + \psi_1(h)(\frac{w_r}{2} - w_d).$$

Case 2: For any $1 \le i \le n$ and $1 \le j \le j' \le m$ such that (i) $j = 1$ or $(j-1, j'') \in P_2$ for some $j'' > j'$ and (ii) each of the arcs of P_2 either completely contains the interval $[j, j']$, or is contained in it, or is disjoint from it,

$$DP(i, i-1; j, j') = \sum_{h=j}^{j'} w_d + \psi_2(h)(\frac{w_r}{2} - w_d).$$

Case 3: For any arc $(j, j') \in P_2$ and $1 \le i \le i' \le n$, we distinguish two subcases: $(i, i') \in P_1$ and $(i, i') \notin P_1$.

In the former subcase,

$$DP(i, i'; j, j') = \min \begin{cases} DP(i+1, i'-1; j+1, j'-1) + (\chi(i,j) + \chi(i',j'))\frac{w_{am}}{2}, \\ DP(i, i'-1; j+1, j'-1) + \frac{3}{2}w_b + w'_d + \chi(i',j')w_m, \\ DP(i+1, i'; j+1, j'-1) + \frac{3}{2}w_b + w'_d + \chi(i,j)w_m, \\ DP(i, i'; j+1, j'-1) + w_r, \\ DP(i, i'-1; j, j') + \frac{1}{2}w_b + w'_d, \\ DP(i+1, i'; j, j') + \frac{1}{2}w_b + w'_d. \end{cases}$$

We remark that the first term of the above formula combines three possibilities, an arc-match, an arc-mismatch with one pair of mismatching bases incurring a cost of $\frac{w_{am}}{2}$, and an arc-mismatch with two pairs of mismatching bases incurring a cost of w_{am}. Notice that $\chi(i,j) + \chi(i',j')$ =0 means an arc-match; $\chi(i,j) + \chi(i',j') = 1$ means an arc-mismatch with cost $\frac{w_{am}}{2}$; and $\chi(i,j) + \chi(i',j')$ = 2 means an arc-mismatch with cost w_{am}.

In the latter case,

$$DP(i, i'; j, j') = \min \begin{cases} DP(i+1, i'-1; j+1, j'-1) + (2 + \psi_1(i) + \psi_1(i'))\frac{w_b}{2} \\ \qquad + (\chi(i,j) + \chi(i',j'))w_m, \qquad \textbf{if } i < i', \\ DP(i, i'-1; j+1, j'-1) + (2 + \psi_1(i'))\frac{w_b}{2} + w'_d + \chi(i',j')w_m, \\ DP(i+1, i'; j+1, j'-1) + (2 + \psi_1(i))\frac{w_b}{2} + w'_d + \chi(i,j)w_m, \\ DP(i, i'; j+1, j'-1) + w_r, \\ DP(i, i'-1; j, j') + w_d + \psi_1(i')(\frac{w_r}{2} - w_d), \\ DP(i+1, i'; j, j') + w_d + \psi_1(i)(\frac{w_r}{2} - w_d). \end{cases}$$

Case 4: For any $1 \le i \le i' \le n$ and $1 \le j \le j' \le m$ such that (i) $(j, j') \notin P_2$, (ii) $j = 1$ or $(j-1, j'') \in P_2$ for some $j'' > j'$, and (iii) each of the arcs of P_2 either completely contains the interval $[j, j']$, or is contained in it, or is disjoint from it, we again consider two subcases.

If j' is free, then

$$DP(i, i'; j, j') = \min \begin{cases} DP(i, i'-1; j, j'-1) + \chi(i',j')w_m + \psi_1(i')\frac{w_b}{2}, \\ DP(i, i'; j, j'-1) + w_d, \\ DP(i, i'-1; j, j') + w_d + \psi_1(i')(\frac{w_r}{2} - w_d). \end{cases}$$

If j' is not free, *i.e.* $j' = u(j')_r$ by the above condition (iii), then

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

Again, it is easy to turn the recurrence relation into a dynamic programming algorithm that computes $DP(i, i'; j, j')$ for all necessary combinations of i, i', j, j', in the ascending order of $j' - j$. The overall running time of the algorithm is $O(n^3m)$. We can also spend an additional $O(n^2m)$ time to recover an optimal alignment by using the standard back-tracing technique. Therefore, we have the follow conclusion.

Theorem 4.1 *Under any score scheme satisfying* $2w_a = w_b + w_r$, *the problem* EDIT(CROSSING, NESTED) *is solvable in* $O(n^3m)$ *time.*

The space complexity of the above algorithm is $O(n^2m)$. This can be improved to $O(nm)$ as follows.

Recall that for any arc in P_1 that has been broken, we would split its cost w_b equally among its two incident bases in the above algorithm. Therefore deleting a base $S_1[i]$ costs $w_d + \psi_1(i)(\frac{w_r}{2} - w_d)$. This idea can also be applied to the base-match and base-mismatch operations. For a base-match, if both bases are free, then the cost is zero; if one of the bases is free, then the cost is $\frac{w_b}{2}$; if both bases are not free, the cost should be $2 \cdot \frac{w_b}{2} = w_b$. For a base-mismatch, if both bases are free, then the cost is still w_m; if one of the bases is free, then the cost is $w_m + \frac{w_b}{2}$; if both bases are not free, the cost is $w_m + 2 \cdot \frac{w_b}{2} = w_m + w_b$. Therefore a base-match or base-mismatch between $S_1[i]$ and $S_2[j]$ will have a cost of $\chi(i,j)w_m + (\psi_1(i) + \psi_2(j))\frac{w_b}{2}$. With this arrangement, we will not need to consider arc-altering, arc-breaking, and arc-removing explicitly since their costs have been incorporated into the costs of base-deletion, base-match, and base-mismatch operations. We can thus derive the following recurrence relation:

For any $1 \le i \le i' \le n$ and $1 \le j \le j' \le m$,

$$DP(i, i'; j, j') = \min \begin{cases} DP(i, i'-1; j, j') + w_d + \psi_1(i')(\frac{w_r}{2} - w_d), \\ DP(i, i'; j, j'-1) + w_d + \psi_2(j')(\frac{w_r}{2} - w_d), \\ DP(i, i'-1; j, j'-1) + \chi(i', j')w_m + (\psi_1(i') + \psi_2(j'))\frac{w_b}{2}, \\ DP(i, i_1 - 1; j, j_1 - 1) + DP(i_1 + 1, i' - 1; j_1 + 1, j' - 1) \\ \quad + (\chi(i_1, j_1) + \chi(i', j'))\frac{w_{am}}{2}, \\ \qquad \textbf{if } i \le i_1, j \le j_1, (i_1, i') \in P_1, \text{and } (j_1, j') \in P_2. \end{cases}$$

A straightforward implementation of this recurrence results in an algorithm with a time complexity of $O(n^2m^2)$. Since we only need to maintain $DP(i+1, i'-1; j+1, j'-1)$ when $(i, i') \in P_1$ and $(j, j') \in P_2$, the space complexity is $O(nm)$.

Moreover, using a technique of Klein [11], one can design an algorithm for the above recurrence relation that runs in $O(n^2m \log m)$ time and $O(n^2m)$ space. The details are omitted here, because we will present a more efficient (approximation) algorithm in the next subsection.

4.2 Approximation Algorithms

Based on the above (exact) algorithm for EDIT(CROSSING, NESTED) under the condition $2w_a = w_b + w_r$, we can construct an approximation algorithm for EDIT(CROSSING, NESTED) under any score scheme. We consider two cases concerning some (arbitrarily) fixed score scheme. Let (S_1, P_1) and (S_2, P_2) be an instance of EDIT(CROSSING, NESTED), where P_1 is (at most) crossing, P_2 is (at most) nested, $|S_1| = n$, and $|S_2| = m$.

Case 1: $2w_a > w_b + w_r$. Suppose that M^* is an optimal alignment for (S_1, P_1) and (S_2, P_2), and M^* has cost d^*. We define a new score scheme in which everything is the same as that in the original score scheme except that the cost of an arc-altering operation is reduced to $\frac{w_b+w_r}{2}$. Using the algorithm in Section 4.1, we can compute an optimal alignment M for (S_1, P_1) and (S_2, P_2) under this new score scheme.

Observe that the alignment M may specify different optimal edit transcripts under the original and the new score schemes, because the original score scheme has a richer set of arc operations. For example, each base deletion in M, where the base is incident on some arc, can be thought of as a subsequent step of some arc-breaking operation under the new score scheme and thus its cost could be computed separately as in Section 4.1. However, under the original score scheme, it should be considered as a part of an arc-altering operation or an arc-removing operation, in addition to being a subsequent step of some arc-breaking operation. In fact, these are the only differences between the optimal edit transcripts specified by M under the new score scheme and the original score scheme. That is, we need only check those arc operations that result in the deletion of some bases. Interestingly, if both end bases of an arc are deleted or if none of them are deleted in an arc operation, then the arc operation implies the same edit operations (with the same cost) under both score schemes. However, if exactly one of the two end bases is deleted in the arc (breaking) operation, then the arc-breaking and base-deletion operations under the new score scheme, with a cost of $\frac{w_r+w_b}{2}$, could be replaced with an arc-altering operation under the original score scheme, with a cost of w_a. Therefore, letting d denote the cost of alignment M under the new score scheme, and d' denote the cost of alignment M under the original score scheme, we have

$$\frac{d'}{d} \leq \frac{2w_a}{w_b + w_r}. \tag{4.1}$$

On the other hand, we can easily compute the cost of alignment M^* under the new score scheme. This could be done by simply replacing each w_a with $\frac{w_b+w_r}{2}$. Therefore, this new cost is at most d^* because $2w_a \geq w_b + w_r$. Since d is the cost of an optimal alignment under the new score scheme, we conclude that

$$d \leq d^*. \tag{4.2}$$

Inequalities (4.1) and (4.2) imply that

$$\frac{d'}{d^*} \leq \frac{2w_a}{w_b + w_r}. \tag{4.3}$$

That is, under the original score scheme, the cost of alignment M computed by the dynamic programming algorithm in Section 4.1 is within $\frac{2w_a}{w_b+w_r}$ times the optimum.

Case 2: $2w_a < w_b + w_r$. In this case, we define a new score scheme by increasing the cost of arc-altering to $\frac{w_b+w_r}{2}$. Similarly, an optimal alignment M under the new score scheme can be computed by the dynamic programming algorithm of Section 4.1 in $O(n^3m)$ time. Suppose that alignment M has cost d under the new score scheme and d' under the original score scheme. Clearly, $d' \leq d$. On the other hand, let M^* be an optimal alignment under the original score scheme and its cost d^*. It is easy to compute the cost of alignment M^*, denoted by d^{**}, under the new score scheme. It follows from the above discussion that $\frac{d^{**}}{d^*} \leq \frac{w_b+w_r}{2w_a}$. Noticing that $d \leq d^{**}$, we have $d' \leq d^{**}$ and thus

$$\frac{d'}{d^*} \leq \frac{w_b + w_r}{2w_a}. \tag{4.4}$$

That is, under the original score scheme, the cost of alignment M computed by the dynamic programming algorithm of Section 4.1 is within $\frac{w_b+w_r}{2w_a}$ times the optimum.

Theorem 4.2 *Under an arbitrary score scheme, problem* EDIT(CROSSING, NESTED) *admits an approximation algorithm with worst-case performance ratio* $\max\left\{\frac{2w_a}{w_b+w_r}, \frac{w_b+w_r}{2w_a}\right\}$. *The algorithm runs in* $O(n^3m)$ *time.*

The following result, which is especially interesting in the comparison of RNA secondary structures, follows from the theorem trivially.

Corollary 4.3 *Under an arbitrary score scheme, problem* EDIT(NESTED, NESTED) *admits an* $O(n^3m)$ *time approximation algorithm with worst-case performance ratio* $\max\left\{\frac{2w_a}{w_b+w_r}, \frac{w_b+w_r}{2w_a}\right\}$.

The time complexity of the above approximation algorithm can be easily improved to $O(n^2m)$ as follows. Let (S_1, P_1) and (S_2, P_2) be an instance of EDIT(CROSSING, NESTED), with $|S_1| = n$ and $|S_2| = m$. First, we observe that it suffices to consider the case that $n = \theta(m)$ (for some appropriately chosen constant factor in the θ notation). If $n \neq \theta(m)$, then we can obtain a trivial alignment of S_1 and S_2 without matching anything whose cost may differ from the true edit distance between (S_1, P_1) and (S_2, P_2) by at most $O(\min\{n, m\})$. This will be a sufficiently good approximation since the edit distance between (S_1, P_1) and (S_2, P_2) is at least $\Omega(\max\{n, m\})$.

Now we describe the new approximation algorithm. Let $\epsilon > 0$ be a sufficiently small constant, whose value will be decided later. Our basic framework is to apply the above approximation algorithm on the instance. Recall that the algorithm performs dynamic programming (as described in Section 4.1, but under an appropriately modified score scheme) on the arc structure of P_2, starting from the smallest *useful* intervals/arcs of sequence S_2, *i.e.* intervals/arcs that are required in the computation that do not contain other useful arcs/intervals, and then moving to larger intervals/arcs. Here, for each useful quadruple (i, i', j, j') that is required in the computation, where $[j, j']$ is an arc or an interval satisfying the conditions of Case 4 in Section 4.1 and $i \leq i'$, we compute a cost for the quadruple in different ways depending on the ratio between $i' - i$ and $j' - j$. If $i' - i \leq \lfloor \epsilon(j' - j) \rfloor$ or $j' - j \leq \lfloor \epsilon(i' - i) \rfloor$, we estimate (an upper bound of) the cost of an optimal solution for the quadruple simply by aligning $S_1[i, i']$ with $S_2[j, j']$ trivially without matching anything (hence specifying an edit transcript that consists of deleting everything in $(S_1[i, i'], P_1[i, i'])$ and then inserting everything in $(S_2[j, j'], P_2[j, j'])$). Observe that this estimation can be carried out in

$O(1)$ time using an appropriate data structure. If $\lfloor \epsilon(j'-j) \rfloor < i'-i < \lfloor (j'-j)/\epsilon \rfloor$, we compute the cost of the quadruple by using the recurrence formulae given in Section 4.1. A crucial idea here is that, when going from an interval $[j, u(j')_l - 1]$ and an arc $(u(j')_l, j')$ to an interval $[j, j']$ on sequence S_2 using the recurrence formula in the second subcase of Case 4 in Section 4.1, we will only consider positions $i'' \in [i, i']$ satisfying both $i'' - i \le \lfloor (u(j')_l - j)/\epsilon \rfloor$ and $i' - i'' \le \lfloor (j' - u(j')_l)/\epsilon \rfloor$. Hence, the range of i'' is limited by $\min\{\lfloor (j' - u(j')_l)/\epsilon \rfloor, \lfloor (u(j')_l - j)/\epsilon \rfloor\}$ and this limitation speeds up the algorithm.

Theorem 4.4 *The above new algorithm for* EDIT(CROSSING, NESTED) *runs in* $O(n^2 m)$ *time and achieves an approximation ratio of* $\max\left\{\frac{2w_a}{w_b+w_r}, \frac{w_b+w_r}{2w_a}\right\}$, *if* ϵ *is so small that* $\frac{1+\epsilon}{1-\epsilon} \le \max\left\{\frac{2w_a}{w_b+w_r}, \frac{w_b+w_r}{2w_a}\right\}$.

PROOF. We first consider the running time of the algorithm. Recall that the algorithm runs in $O(n+m)$ time if $n \le \lfloor \epsilon m \rfloor$ or $m \le \lfloor \epsilon n \rfloor$. So, let us assume that $\lfloor \epsilon m \rfloor < n < \lfloor m/\epsilon \rfloor$. Since the algorithm spends at most $O(n^2 m)$ time on all useful quadruples (i, i', j, j') where $i' - i \le \lfloor \epsilon(j'-j) \rfloor$ or $j' - j \le \lfloor \epsilon(i'-i) \rfloor$, we need only consider the time spent on the other useful quadruples, *i.e.* the ones with $\lfloor \epsilon(j'-j) \rfloor < i' - i < \lfloor (j'-j)/\epsilon \rfloor$. Let $T(j, j')$ denote the total time spent by the algorithm (for computing the costs of such quadruples) on the segment of S_2 covered by a useful interval/arc $[j, j']$ that is required in the overall computation. We prove by induction on $j' - j$ that $T(j, j') = O(n(j'-j)^2)$. Since the total time required by the algorithm on computing the recurrence formulae in Cases 1 – 3 and the first subcase of Case 4 in Section 4.1 is $O(n^2 m)$, we focus on the time spent on the recurrence formula in the second subcase of Case 4. The induction basis holds obviously. Suppose that this is true for any useful interval/arc $[j, j']$ with $j' - j < k$. Consider a useful interval/arc $[j, j']$ with $j' - j = k$. By the induction hypothesis, we have

$$\begin{aligned} T(j, j') &= O(n(u(j')_l - j)^2) + O(n(j' - u(j')_l)^2) + c_1 n(j'-j) \cdot \min\{u(j')_l - j, j' - u(j')_l\}/\epsilon^2 \\ &\le O(n(u(j')_l - j)^2) + O(n(j' - u(j')_l)^2) + c_1 n(u(j')_l - j)(j' - u(j')_l)/\epsilon^2 \\ &\le O(n(j'-j)^2) \end{aligned}$$

In the last term of the first line, c_1 is some appropriate constant, $n(j'-j)/\epsilon$ is the number of intervals $[i, i']$ of S_1 that we have to consider for the given $[j, j']$ of S_2, and $\min\{u(j')_l - j, j' - u(j')_l\}/\epsilon$ is the number of ways of splitting the interval $[i, i']$ considered in our approximation algorithm. The last inequality holds if we choose the constant in big-O notation appropriately (relative to constant c_1 and ϵ). Hence, $T(1, m) = O(nm^2) = O(n^2 m)$, because $\lfloor \epsilon m \rfloor < n < \lfloor m/\epsilon \rfloor$.

Now we prove that the approximation ratio of the algorithm is $\max\{\frac{2w_a}{w_b+w_r}, \frac{w_b+w_r}{2w_a}, \frac{1+\epsilon}{1-\epsilon}\}$, for any ϵ. For any useful quadruple (i, i', j, j'), let $DP'[i, i'; j, j']$ denote the cost computed by the algorithm for the quadruple. Observe that the algorithm follows the dynamic programming algorithm in Section 4.1 faithfully, except when (i) $i' - i \le \lfloor \epsilon(j'-j) \rfloor$ or $j' - j \le \lfloor \epsilon(i'-i) \rfloor$; or (ii) computing the recurrence formula in the second subcase of Case 4. It is easy to see that the first type of deviations result in alignments with costs within $\frac{1+\epsilon}{1-\epsilon}$ times the optima. We argue that the second type of deviations are harmless (*i.e.* they do not affect the result of the algorithm).

To see this, we observe that for any $i'' \in [i, i']$ such that $i'' - i > \lfloor (u(j')_l - j)/\epsilon \rfloor$, the following inequality holds

$$\begin{aligned} & DP'[i, i''-1; j, u(j')_l - 1] + DP'[i'', i'; u(j')_l, j'] \\ \geq \;& DP'[i, \lfloor (u(j')_l - j)/\epsilon \rfloor + i - 1; j, u(j')_l - 1] + DP'[\lfloor (u(j')_l - j)/\epsilon \rfloor + i, i'; u(j')_l, j']. \end{aligned}$$

This is because when $i'' - i > \lfloor (u(j')_l - j)/\epsilon \rfloor$,

$$DP'[i, i''-1; j, u(j')_l - 1] = DP'[i, i''-2; j, u(j')_l - 1] + w_d + \psi_1(i''-1)(\frac{w_r}{2} - w_d),$$

by the definition of the algorithm. Similarly, for any $i'' \in [i, i']$ such that $i' - i'' > \lfloor (j' - u(j')_l)/\epsilon \rfloor$, we have

$$\begin{aligned} & DP'[i, i''-1; j, u(j')_l - 1] + DP'[i'', i'; u(j')_l, j'] \\ \geq \;& DP'[i, i' - \lfloor (j' - u(j')_l)/\epsilon \rfloor - 1; j, u(j')_l - 1] + DP'[i' - \lfloor (j' - u(j')_l)/\epsilon \rfloor, i'; u(j')_l, j']. \end{aligned}$$

Hence, this fast method of computing the recurrence formula in the second subcase of Case 4 does not affect the result of computation, and we may think of the algorithm as computing the recurrence formula faithfully.

Since the cost estimation in the first type of deviations (*i.e.* when $i' - i \leq \lfloor \epsilon(j' - j) \rfloor$ or $j' - j \leq \lfloor \epsilon(i' - i) \rfloor$) involves only arc-removing operations and base-deletions on free bases, we can easily extend the analysis for Theorem 4.2 to show that $DP'[1, n; 1, m]$ is within $\max\{\frac{2w_a}{w_b + w_r}, \frac{w_b + w_r}{2w_a}, \frac{1+\epsilon}{1-\epsilon}\}$ times the optimum. □

5 Some Experimental Results

The source code of the dynamic programming algorithms described in the paper for computing (near) optimal alignments of RNA secondary structures, written in C++, is available in webpage `http://www.csd.uwo.ca/~bma/rna_align/`. We remark that they are very straightforward implementation of the algorithms, and further optimization as well as a graphical user interface are under construction. We have performed a preliminary test on our algorithms for computing EDIT(NESTED, NESTED) on three pairs of real RNAs taken from the RNase P Database [3]: (i) `Alcaligenes eutrophus` and `Streptomyces bikiniensis`, (ii) `Alcaligenes eutrophus` and `Anacystis nidulans`, and (iii) `Anacystis nidulans` and `Streptomyces bikiniensis`. Some comparison of our results and the results from [27] is given below.

Notice that the work in [27] is based on the assumption that a base-pair can only be matched with a base-pair and therefore the matching of a base-pair with two unpaired bases is forbidden. It appears that this assumption in general is good and alignments obtained following the assumption are of good quality [27]. However, sometimes one may have to consider alignments where a base-pair of one RNA is aligned with two unpaired bases of the other. Suppose, for example, that the base-pair involved in the former RNA is (C, G) and the two unpaired unpaired bases in the latter RNA are G and G. Then, the model of [27] would interpret the alignment as the deletion of base-pair (C, G) followed by the insertions of two bases G and G. These edit operations might incur a heavy penalty which may rule out the possibility of such an alignment altogether. However, looking at the sequence level, we can think of the alignment as the result of just one mutation, namely the mutation of C into G. Such a mutation then causes the bond between the base-pair to break. In our edit model, we would charge the alignment the cost of mutating C to G plus the cost of an arc-breaking operation. This is more realistic since mutation of a base in a base-pair should be more expensive than mutation of an unpaired base and less expensive than the deletion of a whole base-pair and then inserting two unpaired bases.

Figures 3, 4, and 5 show the comparison of alignments for the three pairs of RNAs. In each figure, the top alignment was produced by the algorithm in [27] and the bottom one by our algorithm under the score scheme $(w_r, w_a, w_b, w_{am}, w_d, w_m) = (2, 1.75, 1.5, 1.8, 1, 1)$. Indeed, one can find the situations described above in each of these comparisons. The locations of these situations are marked by $*$ in both alignments.

6 Conclusions and Remarks

In this paper, we have studied the similarity comparison between two RNA secondary and tertiary structures. Specifically, we have defined a new edit distance between two RNA secondary and tertiary structures to take into account operations on base-pairs as well as individual bases. By representing RNA secondary and tertiary structures as sets of arcs, we have designed two efficient algorithms, one for EDIT(NESTED, PLAIN) and the other for EDIT(CROSSING, NESTED) under a class of the score schemes where the cost of an arc-altering operation is equal to half the sum of the cost of an arc-breaking operation and the cost of an arc-removing operation. For arbitrary score schemes, we have proved that computing the edit distance is MAX SNP-hard, even if one of the sequences involved is plain. On the positive aspect, we have designed two efficient approximation algorithms with constant worst-case performance ratios. The comparison of alignments produced by our algorithm with alignments produced by an existing algorithm shows that the edit distance defined in this paper is perhaps more plausible in the practice.

However, we remark that it is potentially possible that by introducing arc-breaking and arc-altering operations, one may obtain alignments where the left base of a base-pair in one RNA is aligned with the right base of some base-pair in the other RNA, or a base-pair in one RNA could cross another base-pair in the other RNA. These kinds of alignments would not be realistic, but could perhaps be avoided by appropriately setting the cost of arc operations.

As pointed out by a RECOMB'2001 referee, when comparing two RNA secondary structures, a compensatory base exchange can be taken as a strong signal of structural relationship. To incorporate such information, we may set the cost of an arc-mismatch operation, *i.e.* w_{am}, to 0. That is, we treat it just like an arc-match. We observe that all our results (negative or positive) still hold under this kind of score schemes.

7 Acknowledgment

We thank Paul Kearney and Lusheng Wang for helpful discussions on the topic of similarity comparison between RNA structures. We thank also the anonymous RECOMB01 referees for their informative comments which enhanced the exposition of the paper. Their comments from the biological aspect will also be helpful in our future research on the topic.

References

[1] P. Alimonti and V. Kann. Hardness of approximating problems on cubic graphs. In *Proceedings of the 3rd Italian Conference on Algorithms and Complexity (CIAC'97)*, LNCS 1203, pages 288–298, 1997.

[2] V. Bafna, S. Muthukrishnan, and R. Ravi. Computing similarity between RNA strings. In *Proceedings of 6th Annual Symposium on Combinatorial Pattern Matching (CPM'95)*, LNCS 937, pages 1–16, 1995.

[3] J. Brown. The Ribonuclease P Database. *Nucleic Acids Research*, 27:314, 1999.

[4] M. Brown. Small subunit ribosomal RNA modeling using stochastic context-free grammars. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB'00)*, pages 57–66, 2000.

[5] F. Corpet and B. Michot. RNAling program: alignment of RNA sequences using both primary and secondary structures. *Computer Applications in the Bio-sciences*, 10:389–399, 1994.

[6] P. Evans. *Algorithms and Complexity for Annotated Sequence Analysis.* PhD thesis, University of Victoria, 1999.

[7] M. Goemans and D. Williamson. Improved approximation algorithm for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

[8] D. Goldman, S. Istrail, and C. Papadimitriou. Algorithmic aspects of protein structure similarity. In *IEEE Proceedings of the 40th Annual Conference of Foundations of Computer Science (FOCS'99)*, pages 512–521, 1999.

[9] D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge, 1997.

[10] T. Jiang, G.-H. Lin, B. Ma, and K. Zhang. The longest common subsequence problem for arc-annotated sequences. In *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching (CPM 2000)*, LNCS 1848, Springer, pages 154–165, 2000.

[11] P.N. Klein, Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th Annual European Symposium on Algorithms (ESA)*, LNCS 1461, Springer, pages 91–102, 1998.

[12] H. Lenhof, K. Reinert, and M. Vingron. A polyhedral approach to RNA sequence structure alignment. In *Proceedings of the Second Annual International Conference on Computational Molecular Biology (RECOMB'98)*, pages 153–159, 1998.

[13] V. Levenstein. Binary codes capable of correcting insertions and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

[14] G.-H. Lin, Z.-Z. Chen, T. Jiang, and J. Wen. The longest common subsequence problem for sequences with nested arc annotations. To appear in *Proc. 28th International Colloquium on Automata, Languages, and Programming (ICALP)*, July, 2001, Crete, Greece.

[15] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Science*, 43:425–440, 1991.

[16] Y. Sakakibara, M. Brown, R. Hughey, I. Mian, K. S. der, R. Underwood, and D. Haussler. Recent methods for RNA modeling using stochastic context-free grammars. In *Proceedings of 5th Annual Symposium on Combinatorial Pattern Matching (CPM'94)*, LNCS 807, Springer, pages 289–306, 1994.

[17] D. Sankoff. Simultaneous solution of the RNA folding, alignment, and protosequence problems. *SIAM Journal on Applied Mathematics*, 45:810–825, 1985.

[18] D. Sankoff and J. Kruskal, editors. *Time Wraps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, MA, 1983.

[19] B. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Computer Applications in the Biosciences*, 6:309–318, 1990.

[20] D. Shasha, J. Wang, K. Zhang, and F. Shih. Exact and approximate algorithms for unordered tree matching. *IEEE Transactions on Systems, Man, and Cybernetics*, 24:668–678, 1994.

[21] D. Shasha and K. Zhang. Fast algorithms for the unit cost editing distance between trees. *Journal of Algorithms*, 11:581–621, 1990.

[22] K. Zhang. A constrained edit distance between unordered labeled trees. *Algorithmica*, 15:205–222, 1996.

[23] K. Zhang. Efficient parallel algorithms for tree editing problems. In *Proceedings of 7th Annual Symposium on Combinatorial Pattern Matching (CPM'96)*, LNCS 1075, Springer, pages 361–372, 1996.

[24] K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42:133–139, 1992.

[25] K. Zhang, J. Wang, and D. Shasha. On the editing distance between undirected acyclic graphs and related problems. In *Proceedings of 6th Annual Symposium on Combinatorial Pattern Matching (CPM'95)*, LNCS 937, Springer, pages 395–407, 1995.

[26] K. Zhang. Computing similarity between RNA secondary structures. In *Proceedings of IEEE International Joint Symposia on Intelligence and Systems*, Rockville, Maryland, May 1998, pp. 126-132.

[27] K. Zhang, L. Wang, and B. Ma. Computing similarity between RNA structures. In *Proceedings of 10th Annual Symposium on Combinatorial Pattern Matching (CPM'99)*, LNCS 1645, Springer, pages 281–293, 1999.

[28] M. Zuker. On finding all suboptimal foldings from an RNA molecule. *Science*, 244:48–52, 1989.

```
----------------------------------------------------------------------------------------------------
Alcaligenes eutrophus, Streptomyces bikiniensis, by algorithm in [27]:

(((((((((((((((((( (( ((((((((     )))))))) ))             ((((        (((((  ((((     ))))  (((((
(((((((((((((((((( (( (((( (((     ))) )))) ))             ((((        ((((( (((((     )))) )(((((
AAAGCAGGCCAGGCAACCGCUGCCUGCACCG-CAAGGUGCAGGGGGAGGAAAGUCCGGACUCCACAGGGCAGGGUGU-UGGCUAACAGCCAU-CCACGGC
CGAGCCGGGCGGGCGGCCGCGUGGGG-GUCUUCG-GAC-CUCCCCGAGGAACGUCCGGGCUCCACAGAGCAGGGUG-GUGGCUAACGGCCA-CCCGGGGU
                                                                            **             **

  )))) )((   ((               ( ((( ((((((((    )))))))) )  )))
  )))) )((   ((                 ((( ((((((((    )))))))) )  ))        ((((((       ))))))  ((((((((
AACGUGCGGAAUAGGGCCACAGAGA-CGAGU-CUUGCCGCCGGGUUCGCCCGGCGGGA--AGGGUGAAAC------G-----------------------
GACCCGCGGGACAGUGCCACAGAAAAC-AG-ACCG-CCGGGGACCUCGGUCCUCGG-UAAGG-GUGAAACGGUGGUGUAAGAGACCACCAGCGCCUGAGG
                              **                              *

                    )))))))))))))   ((((                          ((((((    (( (((     ))) ))))))))
   ))))))))         )))))))))))))   ((((    ((((((((    ))))))))  ((((((    ((((((     ))))))))))))
CG---------G-UA---A-CCUCCACCUGGAGCAAUCCCAA-A--------UA-G----------GCAGGCGAU-GA-AGCGGCCCGCU-GAGUCUGCG
CGACUCAGGCGGCUAGGUAAACCCCACUCGGAGCAAGGUCAAGAGGGGACACCCCGGUGUCCCUGCGCGGAUGUUCGAGGGCUGCUCGCCCGAGUCCGCG


    ))))       ((((((((    ))))))))            )))))))  (( (((((  ((((    )))))) ))) ))
    ))))      ((((((( (    ) )))))))           )))))))  (( (((    ((((    ))))   ))) ))
GGUAGGGAGCUGGA-GCCGGCUGGUAACAGCCGGC-CUAGA-GGAAUGGUUGUCACGCACCGUU--UGCCGCAAGGCGGG-CGGGGCGCACAGAAUCCGG
GGUAGACCGCACGAGGCCGGC-GGCAAC-GCCGGCCCUAGAUGGA-UGGCCGUCG-CC-CCG--ACGACCGCGAGGUC--CCGG-GG--ACAGAACCCGG


    ) ))))))))))
    ) ))))))))))
CUUAUCGGCCUGCUUUGCUU
CGUACAGCCCGACUCGUCUG


----------------------------------------------------------------------------------------------------
Alcaligenes eutrophus, Streptomyces bikiniensis, by our algorithm:

(((((((((((((((((( (( ((((((((    )))))))) ))             ((((        ((((( ((((     )))) (((((    )
(((((((((((((((((( (( (((((((      ))))))) ))             ((((        ((((((((((     )))))(((((    )
AAAGCAGGCCAGGCAACCGCUGCCUGCACCGCAAGGUGCAGGGGGAGGAAAGUCCGGACUCCACAGGGCAGGGUGUUGGCUAACAGCCAUCCACGGCAAC
CGAGCCGGGCGGGCGGCCGCGUGGGGGUCUUCG--GACCUCCCCGAGGAACGUCCGGGCUCCACAGAGCAGGGUGGUGGCUAACGGCCACCCGGGGUGAC
                                                                           *             *

))) )((   ((               (((( ((((((((    ))))))))  ))))
))) )((   ((                (( (((((((((    )))))))))  ))        ((((((       ))))))  ((((((((    ))
GUGCGGAAUAGGGCCACAGAGA-CGAGUCUUGCCGCCGGGUUCGCCCGGCGG-GAAGGGUGAAAC------G-----------------------CG---
CCGCGGGACAGUGCCACAGAAAAC-AGACC-GCCGGGGACCUCGGUCCUCGGUAAGG-GUGAAACGGUGGUGUAAGAGACCACCAGCGCCUGAGGCGACU
                           *                             *

               )))))))))))))   ((((                          ((((((    (( (((     ))) ))))))))     )
))))))         )))))))))))))   ((((    ((((((((    ))))))))  ((((((    ((((((     ))))))))))))     )
------G-UA---A-CCUCCACCUGGAGCAAUCCCAA-A--------UA-G----------GCAGGCGAU-GA-AGCGGCCCGCU-GAGUCUGCGGGUAG
CAGGCGGCUAGGUAAACCCCACUCGGAGCAAGGUCAAGAGGGGACACCCCGGUGUCCCUGCGCGGAUGUUCGAGGGCUGCUCGCCCGAGUCCGCGGGUAG


)))       ((((((((    ))))))))            )))))))  (( (((((((((    )))))) ))) ))                 ) )
)))      ((((((( (    ) )))))))           )))))))  (( (((  ((((    ))))   ))) ))                 ) )
GGAGCUGGA-GCCGGCUGGUAACAGCCGGC-CUAGA-GGAAUGGUUGUCACGCACCGUUUGCCGCAAGGCGGG-CGGGGCGCACAGAAUCCGGCUUAUCG
ACCGCACGAGGCCGGC-GGCAAC-GCCGGCCCUAGAUGGA-UGGCCGUCG-CC-CCGACGACCGCGAGGUC--CCGG-GG--ACAGAACCCGGCGUACAG


)))))))))
)))))))))
GCCUGCUUUGCUU
CCCGACUCGUCUG
```

Figure 3: First Comparison of our alignment and that of a previous algorithm.

```
----------------------------------------------------------------------------------------------------
Alcaligenes eutrophus,   Anacystis nidulans, by algorithm in [27]:

  (((( (((  (( (((((( ((( (( ((((((((     ))) ))))) ))             (( ((         ((( ((  ((((     ))
  (((((((  ((( (((((  ((( (( ((((((((     ))) ))))) ))             (((((         ((( (( (((((     ))
--AAAG-CAG--GC-CAGGCA-ACCGCUGCCUGCACCG-CAAGGU-GCAGGGGGAGGAAAGUCCGGACU-CCACAGGGC-AGGG-UGU-UGGCUAACAGC
GCGGGGAAAGGAGGCGAGGC-AGUUGCG-GCUCAGGCUUCG-GUUAUGGGC-UGAGGAAAGUCCGGGCUCCCAAAAGACCAGACUUG-CUGGGUAACGCC
                                                                                       **

))  (((((    )))) )((   ((               (((( (  (((((((     ))))) ))) ))  ))
)) )(((((    )))) )((   ((               (((( (  (((((((     ))))) ))) ))  ))       ((((((       )))
CAU-CCACGGCAACGUGCGGAAUAGGGCCACAGAGACGAG-UCUUGC--CGCCGGG-UUCGCCCGG-CGGGAA--GGGUG-AA---------ACG-----
CA-GUGCGGGUGACCGUGAGGAGAGUGCCACAGAAAC-AUACCGC-CGAUGGCCUGCUU-GCAGGCACAG-GUAAGGGUGCAAGGGUGCGGUAAGAGCGC
  **

                                ))))))))))) ))   ((((      ((((((   (((((         )))))))))))
)))  (((((((    )))))))         ))))))))))))))    (((      ((( ((     (((         )))  )) )))
------------------------C--GGUAACCUCCACCUGG-AGCAAUCCCAAAUAGGCAGGCGAUGAAGC-GGCCC---GCUGAGUCUGCGGGU-A-
ACCAGCAACAUCGAGAGGUGUUGGCUCGGUAAACCCCGGUUGGGAGCAA-GGUGGA--GGGA-CA-AC--GGUUGGUCUUUUACC--UG-UUCCGUUUAU

  ))))      ((((((((    ))))))))           )))) )))  (( (((((((((     ))))))))) ))                 )
  )))       ((((((((    ))))))))           )))  )))   ( ((   ((((     ))))   )) )                  )
--GGGAGCUGGAGCCGGCUGGUAACAGCCGGCCUAGAG-GAAUGGUU-GUCACGCACCGUUUGCCG-CAAGGCGGGCGGGGCGCACAGAAUCCGGCUUAU
GGACC-GCUAGAGGUGGCUAGUAAUAGCCAUCCCAGAGAGA-UAAC-AGCC---C-UC---UGUCUUC--GACA---GA-G-A-ACAGAACCCGGCUUAU

 )))  ))) ))))
 )))) ))))))))
CGGC--CUG-CUUUGCUU-----
GUCCUGCUUUCCCUACUUUAUUU

----------------------------------------------------------------------------------------------------
Alcaligenes eutrophus,   Anacystis nidulans, by our algorithm:

  (((( (((  (( ((((((((( (( ((((((((     ))) ))))) ))             (( ((         ((( (( ((((     ))))
  (((((((  ((( ((((( ((( (( ((((((((     ))) ))))) ))             (((((         ((( (((((((     ))))
--AAAG-CAG--GC-CAGGCAACCGCUGCCUGCACC-GCAAGGU-GCAGGGGGAGGAAAGUCCGGACU-CC-ACAGGGCAGGG-UGUUGGCUAACAGCCA
GCGGGGAAAGGAGGCGAGGCAGUUGCG-GCUCAGGCUUCG-GUUAUGGGC-UGAGGAAAGUCCGGGCUCCCAAAAGACCAGACUUGCUGGGUAACGCCCA
                                                                                      *

 (((((    )))) )((   ((               ((((   ((((((((    )))))) )) ))  ))
)(((((    )))) )((   ((               (((((  (((((((      ))))) )))))  ))       ((((((       ))))))
UCCACGGCAACGUGCGGAAUAGGGCCACAGAGACGAG-UCUU-G-CCGCCGGGUUCGCCCGGC-GGGAA--GGGUG-AA---------A----C------
GUGCGGGUGACCGUGAGGAGAGUGCCACAGAAAC-AUACCGCCGAUGGCCUGCUU-G-CAGGCACAGGUAAGGGUGCAAGGGUGCGGUAAGAGCGCACCA
*

                            ))))))))))) ))   ((((      ((((((   (((((     ))  )))))))))       ) )))
 (((((((    )))))))         ))))))))))))))   (((       (((((    (((           ))) )))))         )))
G-------------------C--GGUAACCUCCACCUGG-AGCAAUCCCAAAUAGGCAGGCGAUGAAGCGGCCCGC--UGAGUCUGCGGGU-A-G-GGAG
GCAACAUCGAGAGGUGUUGGCUCGGUAAACCCCGGUUGGGAGCAAGGU-GGA--GGGACA--ACGGUU-GGUCUU-UUACC-UGUUCCGUUUAUGGACCG

     ((((((((    ))))))))           )))))))  (( (((((((((    ))))))))) ))                 ) )))  )))
     ((((((((    ))))))))           ))) )))   ( (( ((( (      ) ))) )) )                  ) )))) )))
CUGGAGCCGGCUGGUAACAGCCGGCCUAGAG-GAAUGGUUGUCACGCACCGUUUGCCGCAAGGCGGGCGGGGCGCACAGAAUCCGGCUUAUCGGC--CUG
CUAGAGGUGGCUAGUAAUAGCCAUCCCAGAGAGA-UAACAGCC---C-UC-UGU-CUUC---G-ACA-GA-GA--ACAGAACCCGGCUUAUGUCCUGCUU

 ))))
)))))
-CUUUGCUU-----
UCCCUACUUUAUUU
```

Figure 4: Second Comparison of alignments.

```
----------------------------------------------------------------------------------------------------
Anacystis nidulans,   Streptomyces bikiniensis, by algorithm in [27]:

  (((((((( ((( (((((  ((( ((( (((((((    ))) )))) )))             (((((         ((( (((((((     ))))
  (((( (((  (( ((((( (((( ( ( (((((((    ))) )))) ) )             (( ((         ((( (((((((     ))))
GCGGGGAAAGGAGGCGAGGCA-GUUGCGG-CUCAGGCUUCGGUUAUGGG-CUGAGGAAAGUCCGGGCUCCCAAAAGACCAGACUUGCUGGGUAACGCCCA
--CGAG-CCG--GG-CGGGC-GGCCGC-GUGGGGGUCUUCGGAC-CUCCCC-GAGGAACGUCCGGGCU-CCACA-GAGCAGGG-UGGUGGCUAACGGCCA
                           ***                   ***

)(((((    )))) )((   ((               (((((  (((((((    ))))) )))))  ))       ((((((       ))))))  (
)(((((    )))) )((   ((               (((((   ((((((    ))))) ) )))  ))       ((((((       ))))))  (
GUGCGGGUGACCGUGAGGAGAGUGCCACAGAAA-CAUACCGCCGAUGGCCUGCUUGCAGGCACAGGUAAGGGUGCAAGGGUGCGGUAAGAGCGCACCAGC
CCCGGGGUGACCCGCGGGACAGUGCCACAGAAAACAGACCGCC---GGGGACCUCGGUCCU-C-GGUAAGGGUGAAACGGUGGUGUAAGAGACCACCAGC

((( (((    ))) ))))         ))))))))))))))   (((                           (((((        (((
((((((    ))))))))          ))))))))))) ))   ((((    ((((((((    ))))))))  ((((((    ((((((
AAC-AUCGAGAGGU-GUUGGCUCGGUAAACCCCGGUUGGGAGCAAGGU-G-GA-----------G----------GGACA-A--C---GGUUGGUCUUUU
GCCUGAGGCGACUCAGGCGGCUAGGUAAACCCCACUCGG-AGCAAGGUCAAGAGGGGACACCCCGGUGUCCCUGCGCGGAUGUUCGAGGGCUGCUC----

)))    )))))          )))       ((((((((    ))))))))           )))  )))  ((((  ( ((     )) ) ))))
))))))))))))         ))))       ((((((((    ))))))))           ))) )))) (((((  ((((     )))) )))))
ACC----UGUUCCGUUUAUGG-ACCGCUA-GAGGUGGCUAGUAAUAGCCAUCCCAGA-GAGAUAACA-GCC--CUCU--G-UCUUC--GA-C-AGAG-AA
GCCCGAGUCCGCGGGU-A---GACCGC-ACGAGGCCGGCGGCAACGCCGGCCCUAGAUG-GAUGGC-CGUCGCCCCGACGACCG-CGAGGUCCCGGGGA-

              ) )))) ))))))))
              ) )))  ))) ))))
CAGAACCCGGCUUAUGUCCUGCUUUCCCUACUUUAUUU
CAGAACCCGGCGUACAGCC--CGA-CUCGUCUG-----


----------------------------------------------------------------------------------------------------
Anacystis nidulans,   Streptomyces bikiniensis, by our algorithm:
  (((((((( ((( ((((( ((( ((((((((((    ))) )))))))             (((((         ((( (((((((     )))))((
  (((((( (  (( ((((((((( (( (((((((    ))) )))) ))             ((((          ((( (((((((     )))))((
GCGGGGAAAGGAGGCGAGGCAGUUGCGGCUCAGGCUUCGGUUAUGGGCUGAGGAAAGUCCGGGCUCCCAAAAGACCAGACUUGCUGGGUAACGCCCAGUG
--CGAGCC-G--GG-CGGGCGGCCGCGUGGGGGUCUUCGGAC-CUCCCCGAGGAACGUCCGGGCUCC-ACA-GAGCAGGG-UGGUGGCUAACGGCCACCC
                          **                  **

(((    )))) )((   ((               (((((  (((((((    ))))) )))))  ))       ((((((       ))))))  ((((
(((    )))) )((   ((               (((((  ((((((      )))) )))))  ))       ((((((       ))))))  ((((
CGGGUGACCGUGAGGAGAGUGCCACAGAAA-CAUACCGCCGAUGGCCUGCUUGCAGGCACAGGUAAGGGUGCAAGGGUGCGGUAAGAGCGCACCAGCAAC
GGGGUGACCCGCGGGACAGUGCCACAGAAAACAGACCGCC--GGGGAC-CUCG-GUCC-UCGGUAAGGGUGAAACGGUGGUGUAAGAGACCACCAGCGCC

 (((    ))) ))))         ))))))))))))))   (((                           ((((    (  (((         )))
((((    ))))))))         ))))))))) ))))   ((((    ((((((((    ))))))))  ((((((    ((((((         )))
-AUCGAGAGGU-GUUGGCUCGGUAAACCCCGGUUGGGAGCAAGGU--GGA-----------G------------GGAC----AACGGUUGGUCUUUUACC
UGAGGCGACUCAGGCGGCUAGGUAAACCCCACUC-GGAGCAAGGUCAAGAGGGGACACCCCGGUGUCCCUGCGCGGAUGUUCGAGGGCUGCUC----GCC

  )))))          )))       ((((((((    ))))))))           ))) )))  ((((  (((      ))) ))))
)))))))))      ) )))       ((((((((    ))))))))           ))))))) (((((  ((((    )))) )))))
--UGUUC-CGUUUAUGGACCGCUA-GAGGUGGCUAGUAAUAGCCAUCCCAGA-GAGAUAACAGCC--CUCU--GUCUUC---GAC-AGAGAACAGAACCC
CGAGUCCGCGGGUA-G-ACCGC-ACGAGGCCGGCGGCAACGCCGGCCCUAGAUG-GAUGGCCGUCGCCCCGACGACCGCGAGGUCCCGGGGACAGAACCC

      ) )))) ))))))))
      ) )))  ) ))))))
GGCUUAUGUCCUGCUUUCCCUACUUUAUUU
GGCGUACAGCC--C-GACUCGUCUG-----
```

Figure 5: Third Comparison of alignments.