The Comparison of RNA Secondary Structures with Nested Arc-Annotation

Yung-Hsing Peng and Chang-Biau Yang†
Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan
†cbyang@cse.nsysu.edu.tw

Abstract

In recent years, RNA structural comparison becomes a crucial problem in bioinformatics research. Generally, it is a popular approach for representing the RNA secondary structures with arc-annotation sets. Several methods can be used to compare two RNA structures, such as tree edit distance, longest arc-preserving common subsequence (LAPCS) and stem-based alignment. However, these methods may be helpful only for small RNA structures because of their high time complexity. In this paper, we propose a simplified method to compare two RNA structures in $O(mn)$ time, where m and n are the lengths of the two given RNA sequences, respectively. Our method transforms the RNA structures into specific sequences called object sequences, then compares these object sequences to find their common substructures. We test our comparison method with 118 RNA structures obtained from RNase P Database. For any two structures, we try to identify whether they are in the same family by both structure comparison and sequence comparison. In our experiment, we find that our method for comparing RNA structures can yield better hit rates and is faster than the traditional method to compare the RNA sequences. Therefore, our approach for comparing RNA secondary structures is more sensitive in biology and more efficient in time complexity.

1 Introduction

From biological view, the sequence composed of the nucleotides A, G, C and U, is called the *primary structure* of RNA or simply the RNA *sequence*. Nonetheless, due to the hydrogen bonds, an RNA sequence will fold into a *secondary structure*. In Figure 1, there are six kinds of representations for RNA secondary structures.

Among all of the representations, we choose the arc-annotated representation to implement our algo-

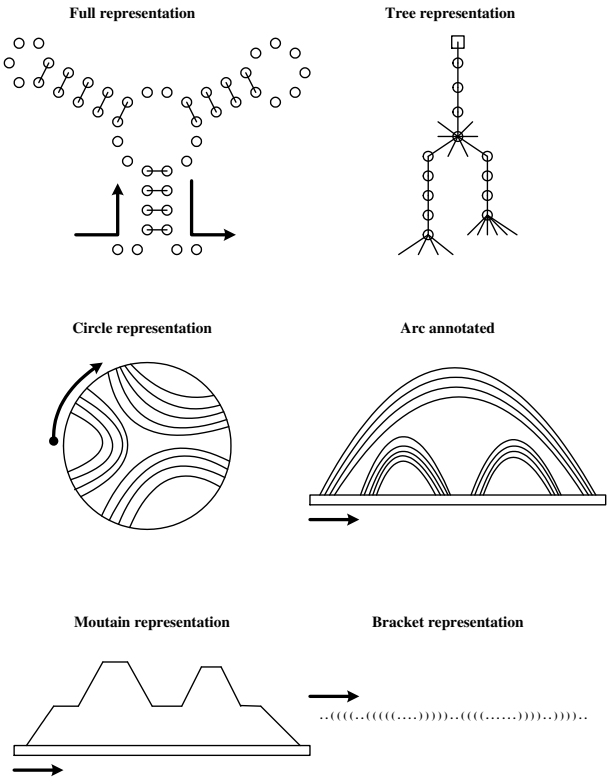

Figure 1: Various kinds of representations for RNA secondary structures.

rithm, because it can easily reveal the nested loops and we can use it to describe our method very fluently. In order to simplify the problems, we will only discuss the RNA structure without pseudoknots, which means the arcs cannot be crossing.

For RNA structure, there are two main kinds of research. One is the structural prediction and the other is the structural comparison. Structure prediction tries to predict the secondary structure of an given RNA sequence. The method for prediction could be based on thermodynamic to find the structure with lowest energy [5, 13], based on the given structure(backbone) to predict the structure of another given sequence [3, 7, 16], or based on sequence similarity to predict local structure [9]. Additional biological or chemical knowledge may be helpful to improve the accuracy.

On the other hand, structure comparison tries to find the common substructure between two given RNA structures, each has its own RNA sequence and base pair set. Common substructure may be defined as the longest arc-preserving common subsequence (LAPCS) between two given arc-annotated sequences [2, 8, 14], the common sub-tree in the tree representation [6, 10, 11, 17], or the stem-based alignment. [15, 19]. As the definitions are different from our view, the computational algorithms are also different from their difficulty and time complexity.

Our research focuses on structure comparison, not structure prediction. In addition, we only consider the structure without pseudoknots. Therefore, please keep in mind that two RNA secondary structures without pseudoknot should be given in advance.

In Section 2, we will introduce some previous algorithms for comparing two RNA structures. Afterwards, we will give a detailed illustration for our method in Section 3 and analysis in Section 4. In Section 5, we show the experimental result. At last, in Section 6 we will come to the conclusions.

2 Previous Work

In this section, we will introduce a previous method to compare two RNA structures, called *stem-based alignment* [18]. This method adopts arc-annotation to represent RNA structures and tries to bind base pairs into single group called *stem*, which are also adopted in our method. Stem-based alignment can be extended to various type [12, 15, 19], therefore we will only make a brief introduction to its core.

In stem-based alignment, the input structures are two sequences and their arc annotation sets. The arcs in the sets cannot be crossing. Stem-based alignment tries to find an alignment that maximizes the score from arc match and base match by breaking some arcs. If one of the given given structures is plain (no arcs), then stem-based alignment will only compute the sequence alignment score.

Stem-based alignment is different from LAPCS since the alignment result may not be a subsequence. Moreover, it is not the same as tree edit distance problem because breaking an arc does not mean to relabel or to delete a node in a tree. In the tree edit distance, relabeling and deletion will change or remove the bases. However, in stem-based alignment, breaking an arc only removes the arc from the arc annotation set, but it does not remove the bases associated with the removed arc.

Given two nested arc-annotated sequences (S_1, A_1) and (S_2, A_2), where S_1 and S_2 are the sequences, A_1 and A_2 are the arc annotation for S_1 and S_2, respectively, let $SeqA(i_1, j_1, i_2, j_2)$ be the sequence alignment score and $SBA(i_1, j_1, i_2, j_2)$ be the stem-based alignment between two structures $S_1[i_1] \sim S_1[j_1]$ and $S_2[i_2] \sim S_2[j_2]$, $i_1 \leq j_1$ and $i_2 \leq j_2$. Then the recusive formula can be given as follows [18].

Suppose $(a_1, b_1) \in A_1$, $(a_2, b_2) \in A_2$, $a_1 \geq i_1$ and $a_2 \geq i_2$, where (a_1, b_1) is an arc in A_1 which is closest to i_1 and (a_2, b_2) is an arc in A_2 which is closest to i_2. We have

$$SBA(i_1, j_1, i_2, j_2) = \max \begin{cases} SBA(i_1, j_1, i_2, j_2) \text{ with } (a_1, b_1) \text{ removed}, \\ SBA(i_1, j_1, i_2, j_2) \text{ with } (a_2, b_2) \text{ removed}, \\ SBA(a_1 + 1, b_1 - 1, a_2 + 1, b_2 - 1) \\ + SBA(i_1, a_1 - 1, i_2, a_2 - 1) \\ + SBA(b_1 + 1, j_1, b_2 + 1, j_2) + match \end{cases}$$

If one of the given structures is a plain structure, in other words the arc annotation set is empty, then $SBA(i_1, j_1, i_2, j_2) = SeqA(i_1, j_1, i_2, j_2)$.

In stem-based alignment, the arcs can also be merged into stem blocks to reduce the number of stems in A_1 and A_2, which is also a good idea in our method. Stem-based alignment can be solved in $O(R_1^2 R_2^2)$ [18] where R_1 and R_2 are the numbers of stems in the given structures. If we modify the input parameters properly, stem-based alignment can maximize the number of arcs which are matched. Figure 2 illustrates the stem-based alignment that maximize the matched arcs by breaking two arcs in structure A and one arc in structure B.

Stem-based alignment can also be extended to edit distance problem by adding other operation rules [15, 19]. In 2002, a general edit distance between two RNA secondary structures [12] was delivered. Up to now, we know that for some editing rules, the optimal solution for the edit distance between a crossing structure and a nested structure can be solve in $O(R_1 R_2 mn)$

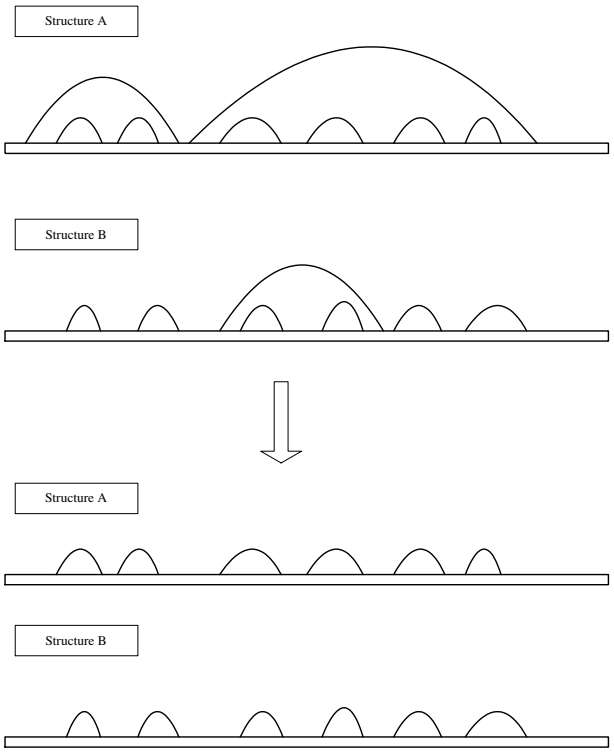

Figure 2: The stem-based alignment.

[19], where $R_1 and\ R_2$ denote the numbers of stems, m and n denote the lengths of the sequences. For different editing rules, the difficulty of the edit problems will also be different. For example, if we adopt the general editing rule, then EDIT(crossing,nested) and EDIT(nested,nested) can only be approximated under a reasonable scoring scheme [12].

3 Our Method

In this section, we will introduce our algorithm for finding the longest common substructure. Our algorithm is based on the concept of LCS and it merges the base pairs into objects. It is not the same as tree alignment because we only consider the object sequence alignment, not the insertion and deletion of every node in the tree. Also, it is different from stem-based alignment because each object can only be compared to another object from outside to inside. Due to these differences, we can prove that the time complexity of our method is $O(mn)$ where m and n denote the lengths of the two RNA sequences. The space complexity of our algorithm is also $O(mn)$. We will introduce our method clearly in the following.

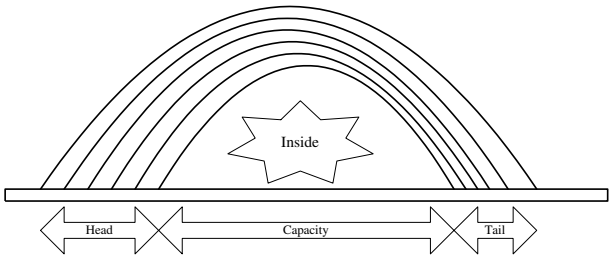

Figure 3: Object analysis.

3.1 Object-based LCS

In our method, an object is defined as a group of base pairs that are nested and consecutive, or very close. Here we use four integers to represent an object, which are the start of the head, the end of the head, the start of the tail and the end of the tail, respectively. From these four integers, we can easily obtain the thicknesses and capacity. Figure 3 shows a simple example. By means of merging close arcs into a single object, we can reduce the number of objects found. This idea is similar to stem, however, here we can set different merging size to fit other situation. If the merging size is set to k, then we merge the nested arcs into an object, satisfying that the distance between any two consecutive arcs is not greater than k. For example, when the merging size is set to 1, every object will actually be a stem. Note that the thicknesses of the head and the tail of the object may be different since the merging size may be greater than 1.

Given an arc-annotated RNA secondary structure, we can scan the RNA sequence and find all object sequences. Figure 4 shows an example that an RNA secondary structure contains two object sequences. Note that object sequences may appear inside another object because the structure is nested. In Figure 4, we can see that the object from 19 to 43 in A_1 contains another object sequence from 20 to 42. Here we name these object sequences according to the start point of the sequence. Therefore, the object sequence scanned from 1 to 60 is named A_1 and that scanned from 20 to 42 is named A_{20}. If the length of the RNA sequence is n, then we will cost $O(n)$ time to find all object sequences by scanning the sequence along the base pairs.

In our method, we try to find the best match between two given object sequences. This problem can also be solved by the concept of dynamic programming, which we call *object-based LCS* (OLCS).

Given two object sequences $A = a_1a_2a_3\cdots a_m$, $B = b_1b_2b_3\cdots b_n$, let $S(i,j)$ denote the best match score of the longest common subsequence between $a_1a_2\cdots a_i$ and $b_1b_2\cdots b_j$ where a_i and b_j represent two objects.

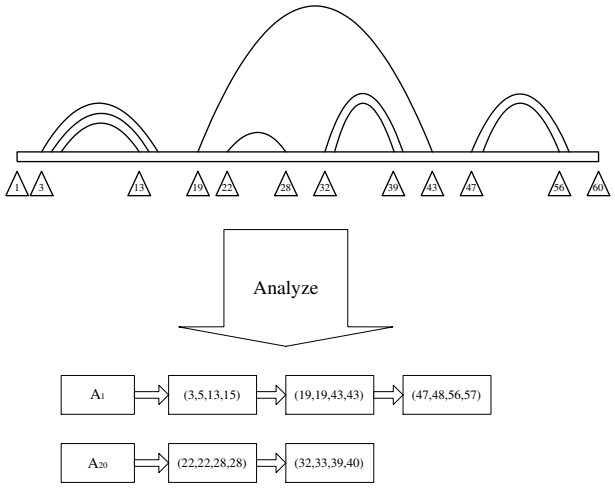

Figure 4: Obtaining the object sequence.

Then the recursive formula for caculating $S(i,j)$ is given as follows.

$$S(i,j) = max \begin{cases} S(i-1,j) \\ S(i,j-1) \\ S(i-1,j-1) + match(a_i,b_j). \end{cases}$$

In above formula, the match function computes the similarity score between a_i and b_j. This function may have more than one definition, which means this function could be flexible. For the method to trace back the OLCS path, we build another tracing table to record the way of tracing back since the match function may be more complicated than it is used in the original LCS problem. With this additional tracing table, we can spare the time for recomputing the match score of two objects and keep the time complexity for tracing in $O(m+n)$.

Given two objects, the match function is the scoring function that determines the similarity between the two objects. In traditional LCS, it is easy to judge if the two objects are the same, because the object here is only a single letter. However, in our case, an object is a structure that may be further nested. Therefore, we modify the scoring function from the traditional LCS, and try to make it fit into our object comparison, called *match function*. We define our match function as follows.

Input: two objects a and b

Output: the match score between a and b

Step 1. Compute the outside score M_O from outside comparison, which compares the size, including the thickness and capacity.

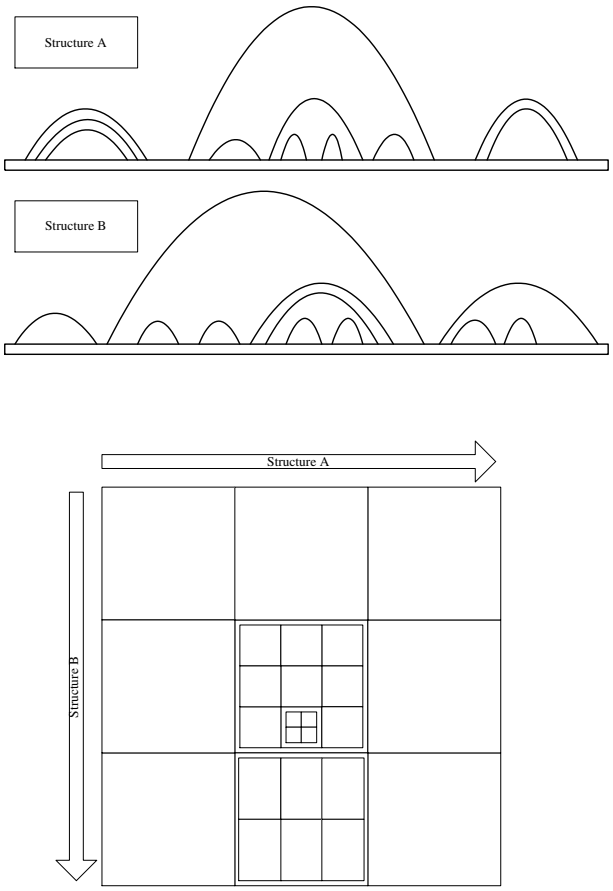

Figure 5: The object-based LCS.

Step 2. If both objects contain inside object sequences (say X in a_i and Y in b_j), then the inside score M_I is determined by $OLCS(X,Y)$. Otherwise, M_I=0, which means there is no inside match score.

Step 3. $M = pM_O + qM_I$, where M is the similarity score of the two objects, p and q are variable parameters.

In the above procedure, we provide a match function that compares two objects from outside to inside. Besides, our function is also flexible in each step, which means we can easily modify each step to deal with other demands in the future.

In order to clarify the relationship between the OLCS and match function in our method, we use Figure 5 as an example. Note that the match function may trigger off another OLCS (in Step 2 above). Nonetheless, when a new OLCS is triggered off, its problem size is actually smaller than the previous one, hence there are some smaller blocks in the scoring table. We shall also use this table to analyze the time complexity of our algorithm later.

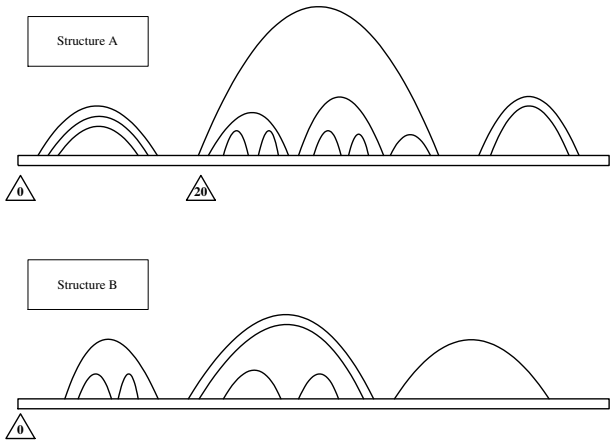

Figure 6: OLCS with different depth.

3.2 OLCS with Different Depths

In the above example, we can see that the depth in both structure *A* and structure *B* is 3. However, in several cases the depths would be different, which implies that *A* may contain *B* or be covered by *B*. If we always perform the OLCS for A_0 and B_0, it means that we will never investigate whether *A* and *B* could contain each other, which may lead to bad solutions. Therefore, when the depth of the two structures are not the same (Without loss of generality, suppose *A* is larger than *B*), we will not only perform OLCS on A_0 and B_0, but also find each object sequence A_i in structure *A* with depth equal to structure *B*, then perform the OLCS on A_i and B_0. Among these solutions, we obtain the solution with the highest OLCS score and treat this solution as the longest common substructure between structures *A* and *B*. It is a heuristic search since object sequence A_i must have the same depth as B_0. However, in this way the time complexity would be kept in $O(mn)$. We will give one more detailed illustration for this idea. In Figure 6, we can see that the depth of A_0 is 3, the depth of A_{20} is 2, the depth of B_0 is 2 and in this example the OLCS for A_{20} and B_0 would be better than the OLCS for A_0 and B_0.

After we apply the OLCS algorithm to find the longest common substructure, the aligned stems can be found. We can treat these aligned stems as the dividing points in the sequence. By doing so, we can split the original two sequences S_A and S_B into two groups of smaller pieces. Figure 7 shows an example of the two structures in Figure 5.

Theoretically, alignment of these smaller pieces could still reveal some message from primary structure comparison. Therefore, we also pay attention to this score, called *Seq_Score*.

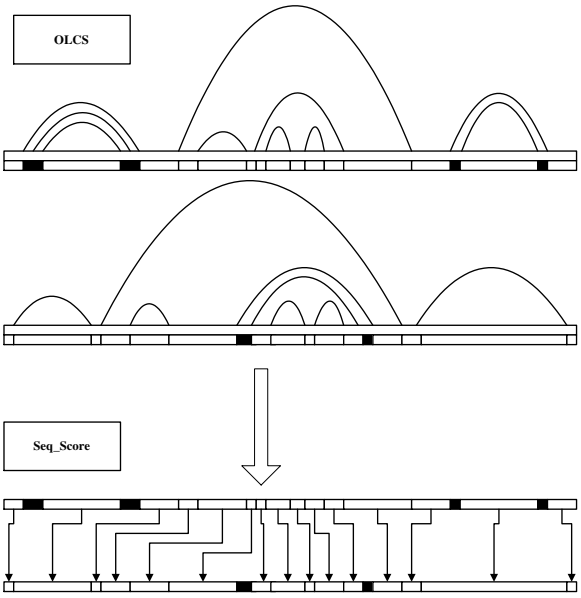

Figure 7: The small pieces of sequences obtained from OLCS.

In our method, *OLCS_Score* could be obtained by OLCS algorithm and *Seq_Score* can be retrieved by the traditional sequence alignment. With this *Seq_Score*, we can identify the structure similarity and sequence similarity individually.

4 Time Complexity and Space Complexity

To analyze the time complexity of our algorithm, we break our whole algorithm into three main steps.

Step 1. Finding the ojbect sequences:
As mentioned before, the time complexity for this step is $O(n)$ where *n* is length of the sequence. For two sequences with lengths *m* and *n*, the time complexity in this step will be $O(m+n)$.

Step 2. Object-based LCS:
In the previous example, we can see the scoring table for OLCS very clearly. Each block in the table may be divided into smaller blocks, which means the match function here invoke a new OLCS. If there is no new OLCS invoked, then the block will not be divided and the time complexity of computation for this block is constant. The total number of blocks would be at most $\frac{mn}{4}$, where *m* and *n* are the lengths of the two sequences, because that rows and columns represent the objects in the two sequences, respec-

tively. For a sequence of length L, it will contain no more than $\frac{L}{2}$ objects, since each object will occupy at least two letters in the sequence if the structure is nested. As a result, we can make a conclusion that there are at most $\frac{mn}{4}$ blocks and the time required for computing this scoring table is $O(mn)$ in the worst case.

For the time complexity of tracing back the solution path in this table, it is obviously $O(m+n)$ since there are at most mn blocks in the table and we perform this trace only once after we build the whole OLCS table.

Step 3. OLCS with different depth:
If structure A is larger than structure B and B has depth d, then we have to find each object sequence A_i in A with depth d and then perform OLCS on A_i and B_0. In this case, it is clear that all A_i's with depth d cannot overlap. In other words, these A_i's are independent object sequences in A. In fact, these A_i's can also be represented as subtrees with height d in A when the tree representation is used. Therefore, performing OLCS on every A_i and B_0 will cost $O(\Sigma m_i n)$ time where m_i is the sequence length of A_i. If all A_i's are independent to each other, then the sequences covered by A_i are also independent to each other. Hence, we can make conclusion that $\Sigma m_i n = O(mn)$. By the above analysis, the time complexity for building the OLCS scoring table in the case with different depth is also $O(mn)$.

Step 4. Additional sequence alignment:
Once the whole OLCS finishes, we then perform the sequence alignment for those smaller sequence pieces produced by OLCS. If we want to align two sequences with lengths m and n, the time complexity would be $O(mn)$. However, if the sequences are breaked into smaller pieces for performing alignment individually, the computation time would be less than $O(mn)$. For the worst case, the time complexity in this step is $O(mn)$.

From Step 1 through Step 4, the time complexity of our algorithm is $O(m+n)+O(mn)+O(mn)+O(mn)=O(mn)$. For the space complexity, in order to store the whole table we need at most mn blocks, therefore the space complexity can be easily verified to be $O(mn)$. The space complexity may be reduced to linear space by implementing the FastLSA algorithm [4] since our algorithm is very similar to the traditional LCS algorithm. Due to the above advantages, our algorithm can use time and memory in an effective way.

5 Experimental Results

In this section, we would first introduce our definition of similarity. By means of this definition, we can easily know how similar two RNAs are, once after we obtain the OLCS score and LCS score. Afterwards, we will show some results about comparing the RNA secondary structures. Here we provide two types of methods for comparison. On one hand, OLCS method will only compare the structural similarity. On the other hand, the traditional LCS will only compute the LCS score for the whole sequences (primary structure) without paying attention to the secondary structure. In this experiment, we can observe the relationship between the sequence and the structure.

Both OLCS and LCS are implemented by C++ and Visual C++ 6.0 is our compiler. We perform our experiment on a PC with AMD Athlon XP 1800+ as proccesor and 256 MB DDR RAM. All of the test data can be obtained from *RNase P Database* [1] on the website http://www.mbio.ncsu.edu/RNaseP/. Among all test data, there exist some pseudoknots. In order to make all test data feasible, we delete the pseudoknot which can be detected in the scanning stage. Our method for deleting the pseudoknot is to ignore the second object if two objects are crossing. By doing so, we can make sure that the input structures are truely nested.

In addition, we provide a simple formula to define the similarity for our experiment. This simple similarity formula is given as follows.

$$Sim = \frac{2 * Func(A,B)}{Func(A,A) + Func(B,B)},$$

where $Func$ in the formula can be OLCS if we want to find the structural similarity. We can also find sequence similarity if we replace the $Func$ by LCS. A and B in the formula means two input RNA secondary structures. In this formula, if two structures are the same then Sim will be 1. Besides, if A contains B or B contains A, Sim will not be 1 because $Func(A,A)+Func(B,B) \geq Func(A,B)$ According to the above reasons, this formula can reflect the similarity in an feasible way.

For the convenience, we shorten some family names in the database and here we provide the mapping table and the relationship of the test data in Table 1 and Figure 8.

To show our results briefly, we would like to give two tables of the distribution of similarity computed by object-based LCS and original LCS. In Table 2, the input sequences are in the same family while in Table 3, the input sequences are in different families. Here we set the merging size to 5 when we perform our OLCS algorithm.

Table 1: Name mapping table of test data.

	Modified Family Name	Original Family Name	Number of sequences
1	Bacteroide	Bacteroides	9
2	Crena	Crenarchaeal	8
3	Plancto	Planctomycete	9
4	Spiroch	Spirochaete	7
5	EuryTA	Euryarchaeal Type A	28
6	Nuclear	Nuclear	57

Table 2: Distribution of similarity in Bacteroide(9 seqs).

	Similarity	Structure(OLCS)	Sequence(LCS)
1	$0.9 \sim 1.0$	9	0
2	$0.8 \sim 0.9$	15	0
3	$0.7 \sim 0.8$	12	23
4	$0.6 \sim 0.7$	0	12
5	$0.5 \sim 0.6$	0	1
6	$0.4 \sim 0.5$	0	0
7	$0.3 \sim 0.4$	0	0
8	$0.2 \sim 0.3$	0	0
9	$0.1 \sim 0.2$	0	0
10	$0.0 \sim 0.1$	0	0
	Similarity	Structure(OLCS)	Sequence(LCS)
1	$0.79 \sim 0.80$	1	0
2	$0.78 \sim 0.79$	1	2
3	$0.77 \sim 0.78$	2	2
4	$0.76 \sim 0.77$	0	1
5	$0.75 \sim 0.76$	1	1
6	$0.74 \sim 0.75$	4	3
7	$0.73 \sim 0.74$	1	5
8	$0.72 \sim 0.73$	1	2
9	$0.71 \sim 0.72$	1	7
10	$0.70 \sim 0.71$	0	0
	Similarity	Structure(OLCS)	Sequence(LCS)
1	$0.69 \sim 0.70$	0	0
2	$0.68 \sim 0.69$	0	1
3	$0.67 \sim 0.68$	0	1
4	$0.66 \sim 0.67$	0	1
5	$0.65 \sim 0.66$	0	1
6	$0.64 \sim 0.65$	0	1
7	$0.63 \sim 0.64$	0	3
8	$0.62 \sim 0.63$	0	3
9	$0.61 \sim 0.62$	0	1
10	$0.60 \sim 0.61$	0	0

Table 3: Distribution of similarity between Nuclear(57 seqs) and EuryTA(28 seqs).

	Similarity	Structure(OLCS)	Sequence(LCS)
1	$0.9 \sim 1.0$	0	0
2	$0.8 \sim 0.9$	0	0
3	$0.7 \sim 0.8$	15	0
4	$0.6 \sim 0.7$	167	860
5	$0.5 \sim 0.6$	193	733
6	$0.4 \sim 0.5$	424	3
7	$0.3 \sim 0.4$	563	0
8	$0.2 \sim 0.3$	232	0
9	$0.1 \sim 0.2$	2	0
10	$0.0 \sim 0.1$	0	0
	Similarity	Structure(OLCS)	Sequence(LCS)
1	$0.79 \sim 0.80$	0	0
2	$0.78 \sim 0.79$	0	0
3	$0.77 \sim 0.78$	0	0
4	$0.76 \sim 0.77$	2	0
5	$0.75 \sim 0.76$	1	0
6	$0.74 \sim 0.75$	1	0
7	$0.73 \sim 0.74$	0	0
8	$0.72 \sim 0.73$	0	0
9	$0.71 \sim 0.72$	6	0
10	$0.70 \sim 0.71$	5	0
	Similarity	Structure(OLCS)	Sequence(LCS)
1	$0.69 \sim 0.70$	2	0
2	$0.68 \sim 0.69$	2	0
3	$0.67 \sim 0.68$	27	1
4	$0.66 \sim 0.67$	24	3
5	$0.65 \sim 0.66$	23	9
6	$0.64 \sim 0.65$	21	53
7	$0.63 \sim 0.64$	11	104
8	$0.62 \sim 0.63$	20	255
9	$0.61 \sim 0.62$	20	227
10	$0.60 \sim 0.61$	17	208

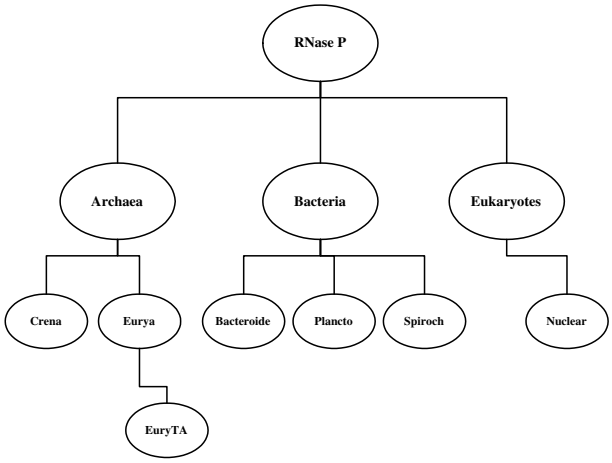

Figure 8: Relationship of the test data.

In above tables, we can see the distribution of similarity computed by two kinds of views. In these tables, we found an interesting trend. For LCS, the bound of family identity seems to be very close to 0.65 in general. Therefore, we search the bound from 0.63 to 0.67 individually and finally we found that 0.64 is the best bound for LCS. As for OLCS, we also search for the bound from 0.73 to 0.77 and finally we set the bound at 0.74. Here we have $\binom{118}{2}$=6903 cases since we have 118 sequences in the sample. We list the hit rates of OLCS in Table 4 and that of LCS in Table 5. OLCS completes these 6903 comparisons in 4 seconds while LCS completes these comparisons in 17 seconds. It is because OLCS merges the arcs into objects and makes the problem size smaller. The total hit rates of OLCS and LCS are $\frac{5303}{6903}$=0.7682 and $\frac{4996}{6903}$=0.7237, respectively.

6 Conclusion

In this section, we will make some conclusions about our method and experimental results. From algorithmic view, OLCS is an algorithm for finding common structures between two RNA secondary structures in $O(mn)$ time. Although OLCS may not be very accurate as tree alignment or LAPCS due to its low time complexity, it still provides another good choice when we want to compare two RNA structures. In addition, OLCS is easy to be implemented because it is modified from LCS and it is applicable because of its low time complexity.

In our experimental results, we showed the structural similarity computed by OLCS and the sequence similarities computed by LCS. Generally, the value range of structural similarity is larger than that of sequence similarity, which makes the structural similarity more sensitive. In the tables of hit rates, we can see that in some inter-family identification, both OLCS and LCS has very low hit rates (*Bacteroide* vs *Plancto*, *Bacteroide* vs *Spiroch*). In thess cases, we find that these two families belongs to the same parent family. As a result, it is reasonable that these two families may have some substructures or subsequenes in common. That is, the similarity between these two families could be a little bit high and it makes the identification more difficult.

We also find that for the intra-identification of *Nuclear*, OLCS and LCS both has very low hit rates. In this case, we find that the number of sequences in the *Nuclear* is much more than that in other families, which implies this family may contain some child families. As for the family classification (searching for the child families), we leave this work to the future because at this time we have no idea about how to split the parent correctly.

If we set the bounds of OLCS and LCS to their own best bound, we can see that in Table 4 and Table 5, OLCS only loses 4 hit rates among all 21 hit rates. It implies that comparing the RNA secondary structures is more sensitive than comparing the primary structures (sequences). In addition, in our experiments we find that OLCS is even faster than LCS. Therefore, we can make a conclusion that our method for comparing RNA secondary structures is more sensitive in biology and more efficient in time complexity.

References

[1] "RNase P Database." http://www.mbio.ncsu.edu /RNaseP/.

[2] J. Alber, J. Gramm, J. Guo, and R. Niedermeier, "Computing the similarity of two sequences with nested arc annotations," *Theoretical Computer Science*, pp. 337–358, 2004.

[3] M. Andronescu, A. P. Fejes, F. Hutter, H. H. Hoos, and A. Condon, "A new algorithm for RNA secondary structure design," *Journal of Molecular Biology*, pp. 607–624, 2004.

[4] K. Charter, J. Schaeffer, and D. Szafron, "Sequence alignment using FastLSA," *Mathematics and Engineering Techniques in Medicine and Biological Sciences*, pp. 239–245, 2000.

[5] Y. Ding and C. E. Lawrence, "A statistical sampling algorithm for RNA secondary structure prediction," *Nucleic Acids Research*, pp. 7280–7301, 2003.

Table 4: Hit rate of family identity for OLCS with 0.74 as bound(5363 hits in 6903 cases).

	Bacteroide	Crena	Plancto	Spiroch	EuryTA	Nuclear
Bacteroide	0.917	1.000	0.123	0.190	0.937	1.000
Crena	-	0.679	1.000	1.000	0.549	0.980
Plancto	-	-	1.000	0.524	0.992	1.000
Spiroch	-	-	-	1.000	0.776	1.000
EuryTA	-	-	-	-	0.627	0.997
Nuclear	-	-	-	-	-	0.336

Table 5: Hit rate of family identity for LCS with 0.64 as bound(4996 hits in 6903 cases).

	Bacteroide	Crena	Plancto	Spiroch	EuryTA	Nuclear
Bacteroide	0.806	0.611	0.198	0.413	0.714	0.965
Crena	-	0.679	0.889	0.714	0.638	0.971
Plancto	-	-	1.000	0.429	0.615	0.875
Spiroch	-	-	-	0.667	0.837	0.952
EuryTA	-	-	-	-	0.627	0.958
Nuclear	-	-	-	-	-	0.318

[6] S. Dulucq and L. Tichit, "RNA secondary structure comparison: exact analysis of the zhang-shasha tree edit algorithm," *Theoretical Computer Science*, pp. 471–484, 2003.

[7] S. R. Eddy, "A memory-efficient dynamic programming algorithm for optimal alignment of a sequence to an RNA secondary structure," *BMC Bioinformatics*, pp. 3–18, 2002.

[8] P. Evans, "Algorithms and complexity for annotated sequence analysis." citeseer.ist.psu.edu /evans99algorithms.html, Department of Computer Science, University of Victoria, 1999.

[9] I. L. Hofacker, M. Fekete, and P. F. Stadler, "Secondary structure prediction for aligned RNA sequences," *Santa Fe Institute Working Papers*, 2001.

[10] I. L. Hofacker, W. Fontana, P. F. Stadler, L. S. Bonhoeffer, M. Tacker, and P. Schuster, "Fast folding and comparison of RNA secondary structures," *Santa Fe Institute Working Papers*, 1993.

[11] J. Jansson and A. Lingas, "A fast algorithm for optimal alignment between similar ordered trees," *Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching*, pp. 232–240, 2001.

[12] T. Jiang, G.-H. Lin, B. Ma, and K. Zhang, "A general edit distance between RNA structures," *Journal of Computational Biology*, pp. 371–388, 2002.

[13] V. Juan and C. Wilson, "RNA secondary structure prediction based on free energy and phylogenetic analysis," *Journal of Molecular Biology*, pp. 935–947, 1999.

[14] G.-H. Lin, Z.-Z. Chen, T. Jiang, , and J. Wen, "The longest common subsequence problem for sequences with nested arc annotations," *International Colloquium on Automata, Languages and Programming*, pp. 444–455, 2001.

[15] G.-H. Lin, B. Ma, and K. Zhang, "Edit distance between two RNA structures," *Research in Computational Molecular Biology*, pp. 211–220, 2001.

[16] T. Liu and B. Schmidt, "Parallel RNA sequence-structure alignment," *High Performance Computational Biology*, 2004.

[17] J. T. Wang, B. A. Shapiro, D. Shasha, K. Zhang, and K. M. Currey, "An algorithm for finding the largest approximately common substructures of two trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998.

[18] M.-Y. Wu, C.-B. Yang, and K.-S. Huang, "RNA secondary structure alignment based on stem representation," *Proceedings of the 21st Workshop on Combinational Mathematics and Computation Theory*, Taichung, Taiwan, pp. 60–69, 2004.

[19] K. Zhang, L. Wang, and B. Ma, "Computing similarity between RNA structures," *Combinatorial Pattern Matching*, pp. 281–293, 1999.