A Memory Efficient Method for Structure-Based RNA Multiple Alignment

Daniel DeBlasio, Jocelyne Bruand, and Shaojie Zhang

Abstract—Structure-based RNA multiple alignment is particularly challenging because covarying mutations make sequence information alone insufficient. Existing tools for RNA multiple alignment first generate pairwise RNA structure alignments and then build the multiple alignment using only sequence information. Here we present PMFastR, an algorithm which iteratively uses a sequence-structure alignment procedure to build a structure-based RNA multiple alignment from one sequence with known structure and a database of sequences from the same family. PMFastR also has low memory consumption allowing for the alignment of large sequences such as 16S and 23S rRNA. The algorithm also provides a method to utilize a multi-core environment. We present results on benchmark data sets from BRAliBase, which shows PMFastR performs comparably to other state-of-the-art programs. Finally, we regenerate 607 Rfam seed alignments and show that our automated process creates multiple alignments similar to the manually-curated Rfam seed alignments. Thus, the techniques presented in this article allow for the generation of multiple alignments using sequence-structure guidance while limiting memory consumption. As a result, multiple alignments of long RNA sequences, such as 16S and 23S rRNAs, can easily be generated locally on a personal computer.

Index Terms—RNA multiple alignment, RNA secondary structure, RNA sequence-structure alignment, iterative alignment

1 Introduction

A high quality multiple alignment of RNA sequences is crucial for RNA homology search [1], structure analysis [2] and discovery [3], [4]. Even though methods for multiple alignments of DNA and protein sequences are well studied [5], [6], [7], [8], structure-based RNA multiple alignment is still an open problem.

Aligning two RNA sequences with consideration of the structural information comes as an extension of the RNA structure prediction studies [9], [10], [11]. When aligning two RNA sequences, we can consider three problems and associated methods depending on whether or not we have the structural information for these RNA sequences [12], [13]. Without considering the pseudo-knots in the RNAs, all these problems can be solved in polynomial time. First, it is possible to align two RNA sequences without any structural information and to predict their common secondary structure [14]; this is the RNA *sequence-sequence* alignment problem. Another possible input is two RNA sequences with the structural profile or other structural information for only one of the sequences; this is the RNA *sequence-structure* alignment problem. Several methods have been developed for this problem and are used for finding RNA homologs. FastR [15] and PFastR [1] use guide trees and dynamic programming to globally align a sequence or profile to a given target sequence. CMSEARCH [16] and RSEARCH [17] use covariance models to supplement dynamic programming for their alignment procedure. Finally, the structural information can be given for both RNA sequences, allowing us to find common motifs between two RNA structures [13]; this is the RNA *structure-structure* alignment problem.

Much work has already been done on the structure-based RNA multiple alignment problem. Most of these RNA multiple alignment methods (PMmulti [18], MARNA [19], Stemloc [20], STRAL [21], and LARA [12]) use pairwise RNA alignments (sequence-sequence or structure-structure) for all sequences and then combine these alignments into a multiple alignment using T-Coffee [7] or other progressive strategies. However, in the case of sequence-sequence alignment, these methods predict the RNA structure or pairing probabilities from scratch at the expense of RNA structure accuracy. For instance, LARA either takes in the annotated structure or predicts the structure of the input sequences; it then uses a connection graph and integer linear programming to create pairwise alignments. The scores of these pairwise alignments is then fed into T-Coffee to generate a multiple alignment. MARNA approaches the problem by using a scoring matrix for each pairwise alignment and focuses on the structure; if the structure is unavailable, MARNA predicts the structure using one of several known prediction tools. Those pairwise align-

- *D. DeBlasio and S. Zhang are with the Department of Electrical Engineering and Computer Science at the University of Central Florida, Orlando, FL 32816, USA.*
  E-mail: {deblasio,shzhang}@eecs.ucf.edu
- *J. Bruand is with the Bioinformatics and Systems Biology Graduate Program at the University of California, San Diego, La Jolla, CA 92093, USA.*
  E-mail: jocelyn@ucsd.edu

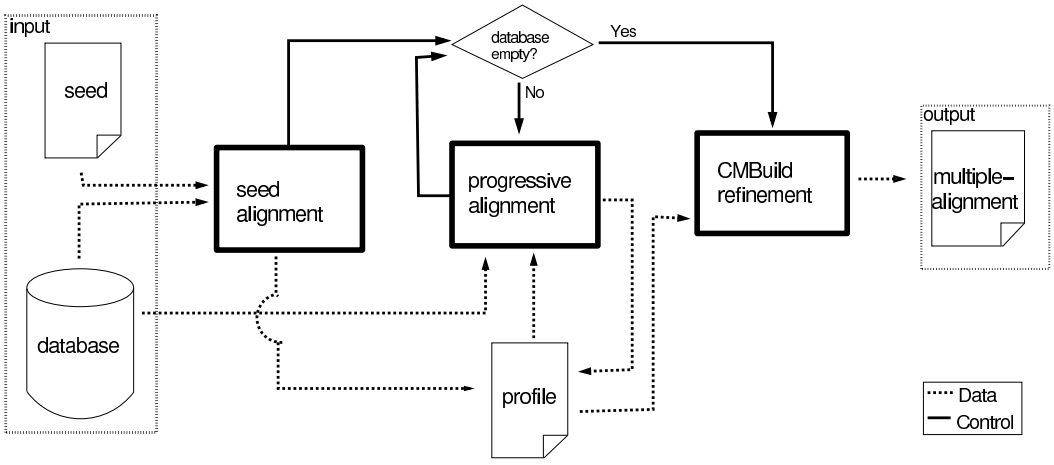

Fig. 1. Overview of PMFastR. The three major steps are highlighted in the solid square boxes, while the input and output of the program are represented in dashed boxes. After generation of a seed alignment, the algorithm progressively aligns the rest of the input data set. Finally, we run the CMBuild refinement program to improve the unpaired gap regions in the alignment.

ments are then also used as input for T-Coffee. This approach is limited to the sequence length because of the running time ($O(n^2l^4) + O(n^3l^2)$, where n is the number of sequences, l is the length). PMmulti uses base-pairing probability matrices for alignment procedure. Again the running time is prohibitive, on the order of $O(l^4)$. Stemloc utilizes a Stochastic Context-Free Grammar (SCFG) and dynamic programming to both align two sequences and predict their structures simultaneously. STRAL uses a heuristic to reduce the sequence-structure alignment problem to a pure sequence based alignment problem.

On the other hand, RNA structure-structure alignment is not feasible on very long RNA sequences, such as 16S rRNA and 23S rRNA. RNAforester [22] performs a structure-based RNA multiple alignment in $O(s^2n^2d^2)$ (where s is the length of the sequence, n is the number sequences and d represents the maximum degree of the tree representing the RNA structure). The trees representing the RNA secondary structures used in RNAforester are very similar to the one presented in this paper, but RNAforester requires that each sequence have a known associated secondary structure. In contrast, our proposed method requires only one RNA sequence with known secondary structure and a database of other unaligned sequences as input.

The RNA sequence-structure alignment strategy is also used to build a multiple alignment. Eddy and Durban [2] accomplish this by using a covariance model, but their algorithm requires an initial multiple alignment as input. This initial alignment would need to be hand-curated or generated by some other alignment method.

Databases of multiple RNA alignments, such as Rfam [23], maintain very high quality alignments, which are obtained by integration of multiple sources and manual curation. We assume that these databases have the correct alignments and, therefore, can be used for a baseline comparison as a mean to assess how well our algorithm is performing. This paper shows that the proposed algorithm can produce comparable results without manually curating the alignments.

In this paper, we present the Profile based Multiple Fast RNA alignment (PMFastR) algorithm. An overview of the method is presented in Figure 1. PMFastR constructs a structure-based RNA multiple alignment from scratch while using a relatively small amount of memory and can be used for long RNA sequences such as 16S rRNA or 23S rRNA sequences. The algorithm consists of three major steps. First, a sequence-structure alignment of an RNA sequence from the database to the original structure is generated; this step outputs an aligned profile of two sequences with structure information. Second, RNA sequence from the database is then aligned to the output profile and obtain a new alignment profile. This can be repeated iteratively until all of the RNA sequences of the input database are aligned. Finally, the CMBuild refinement protocol is run to improve the unpaired gap regions in the alignment. In the Methods Section, we present the algorithm itself and details on all of the major improvements over FastR. In the Results Section, several benchmarking tests on PMFastR with other multiple RNA alignment tools are presented using BRAliBase (version 2.1) [24] as test data sets, which are extracted from Rfam database. We also regenerate all Rfam hand-curated seed alignments (version 8.1) [23] automatically with comparable results.

2 METHODS

The method presented here follows the underlying alignment procedure used in FastR [15] and PFastR [1]. This algorithm utilizes the underlying structure of an RNA sequence to create a tree and align a sequence to that tree by a simple traversal. The sequence-structure alignment problem can then be solved in $O(mn^2)$ time and memory using this approach.

We are able to improve upon the memory consumption as well as the time complexity by utilizing banding. Additionally, we make several other improvements to the basic algorithm to further reduce the running time. Finally, we modify the algorithm to follow a progressive paradigm in order to create a multiple alignment. In this section, we present the original tree based alignment method, followed by a detailed description of each step to expand on the original ideas.

2.1 The Alignment Procedure

By assuming that no pseudoknot exists in an RNA structure, we can create a tree of the structure of that RNA sequence. Let M be the set of all such base pairs. For each base pair $(i, j) \in M$, there is a unique $parent$ base pair (i', j') such that $i' < i < j < j'$, and there is no base pair (i'', j'') such that $i' < i'' < i$ or $j < j'' < j'$. Additionally there must be a base pair (i_r, j_r) where no such parent exists, this node is the root of the tree and all other nodes are connected to their direct parents [25], [26].

Two issues must be resolved in order for the tree to be easily traversed: (1) the tree must represent the entire RNA sequence (including the unpaired bases), and (2) the tree must be binary. To satisfy both of these conditions we follow the binarization procedure presented in FastR [15]. All of the nodes in M are labeled as solid nodes. We introduce a new type of node, the dotted node, to the tree. First, we add dotted nodes at any splitting point in the tree, each solid node that has more than one child will instead become the parent of a dotted node, this node will then have two children, if needed one of these will be another dotted node with two children. To include all bases in the tree, dotted nodes are added at any point where an unpaired base should exist. Because unpaired bases should only exist between stacks in the sequence (or outside the root pair) they have a unique location in the tree. We call M' the new set of nodes (both solid and dotted) from the newly formed tree. Since the nodes in the tree represent at most two locations in the original sequence, we can then try and match those two locations to the target sequence. The entire alignment procedure is shown in Figure 2(a). For a profile input each node is represented by a position specific scoring matrix (PSSM). Because $|M'|$ is bounded by the size of the input profile, it is easy to see how this original algorithm is $O(mn^2)$ in both time and space, any location in the search sequence is considered for a match at any node.

2.2 Banding

Because PMFastR is performing a global alignment, we can assume that the location of matching bases between the profile and the target sequence are similar. In particular, we assume that it is within some banding constant of the original location. Once this assumption has been established, the search space is limited to these bounds. This allows for a reduction in running time since we do not examine locations with a very low likelihood of having a match. The space consumption is also reduced since we only need to store the alignment values for the locations within the banding region.

A banding variable ($band$) is defined and this value can be adjusted depending on the type and length of sequence being examined, as well as the precision of the result desired. Thus, for any node v in the tree, the algorithm only needs to examine the locations in the query where the corresponding base pair or unpaired base might exist. For example, let v be a node in the binarized tree as described above where v represents the base pair at (l_v, r_v). The algorithm looks for the corresponding base pair in the query and only examines the potential pairing sites (i, j) in the query where $l_v - band \leq i \leq l_v + band$ and $r_v - band \leq j \leq r_v + band$.

This banding leads to the assertion that any potential pairings outside of the bounds are never assigned a score. Since the banding constant is given at the beginning of the algorithm, only the space necessary to store results within the banding bounds needs to be allocated and stored. If there is a reference to a location outside those bounds the initialization value is returned. The running time and space complexity of this algorithm is then reduced from $O(n^2 * m)$ to $O(band^2 * m)$, where n is the length of the target sequence and m is the number of nodes in M' which are bounded by the length of the profile. Figures 2(b) and 2(c) show the methods used for mapping into the new bounded array. We can see that these procedures have a constant running time, thus the effect on running time is not significant.

Since the sequences to be aligned are from the same family, we can expect that they are of similar length. However, in some cases, lengths can vary significantly within the same family. Additionally, as the profile grows in height it also grows in width due to insertions from the new sequences. This is a problem for banding. Because we only examine a small portion of the profile, we need to verify this portion still has potential to give a good alignment. The most straight-forward adjustment is a change in the banding parameter. By increasing this value,

procedure PAln
(* M is the set of base pairs in RNA profile R. M' is the augmented set. *)
for all nodes $v \in M'$,
all intervals (i, j), $l_v - band \leq i \leq l_v + band$ and $r_v - band \leq j \leq r_v + band$
if $v \in M$

$$value = \max \begin{cases} mapRetrieve(\text{child}(v), i+1, j-1) + \delta(l_v, r_v, t[i], t[j]), \\ mapRetrieve(v, i, j-1) + \gamma('-', t[j]), \\ mapRetrieve(v, i+1, j) + \gamma('-', t[i]), \\ mapRetrieve(\text{child}(v), i+1, j) + \gamma(l_v, t[i]) + \gamma(r_v, '-'), \\ mapRetrieve(\text{child}(v), i, j-1) + \gamma(l_v, '-') + \gamma(r_v, t[j]), \\ mapRetrieve(\text{child}(v), i, j) + \gamma(l_v, '-') + \gamma(r_v, '-'), \end{cases}$$

else if $v \in M' - M$, and v has one child

$$value = \max \begin{cases} mapRetrieve(\text{child}(v), i, j-1) + \gamma(r_v, t[j]), \\ mapRetrieve(\text{child}(v), i, j) + \gamma(r_v, '-'), \\ mapRetrieve(v, i, j-1) + \gamma('-', t[j]), \\ mapRetrieve(v, i+1, j) + \gamma('-', t[i]), \end{cases}$$

else if $v \in M' - M$, and v has two children

$$value = \max_{i \leq k \leq j} \{ mapRetrieve(\text{left\_child}(v), i, k-1) + mapRetrieve(\text{right\_child}(v), k, j) \}$$

end if
$mapSet(v, i, j, value)$
end for

(a)

procedure mapRetrieve(v, i, j)
(* i and j are the global position in the table assuming that banding is not used. *)
if i & j are within the $band$ of l_v and r_v
$i_t = i - l_v + band$
$j_t = j - r_v + band$
return A[i_t, j_t, v]
else
return initialization value for (i, j, v)
end if

(b)

procedure mapSet($v, i, j, value$)
if i & j are within the $band$ of l_v and r_v
$i_t = i - l_v + band$
$j_t = j - r_v + band$
A[i_t, j_t, v] = $value$
end if

(c)

Fig. 2. (a) An algorithm for aligning an RNA profile R with m columns against a database string t of length n. The query consensus structure M was *Binarized* to obtain M'. Each node v in the tree corresponds to a base pair $(l_v, r_v) \in M'$. (b) and (c) The mapping functions that make the transition to a memory-saving banding array. It is assumed that the array A is of size $n * n * m$ while the new banding array is of size $band * band * m$.

we increase the probability that the proper alignment traceback will stay within the allocated area. The trade-off here is that as this value is increased, the benefit gained from using banding is lost. In fact, because of the properties of banding, if the constant is raised too high, it can actually be detrimental to both running time and memory bounds.

Another solution is to judge each column for quality and hide columns that are not beneficial to the alignment during the procedure. This is achieved by assigning a quality metric to each column. The *quality score* is defined to be the percentage of a column that has a sequence letter assigned to it. Let us note at this point that we only hide columns that do not contain structural information. We then identify all columns that have a *quality score* below some threshold and remove them from the profile. After running the alignment procedure, we add the columns back into the profile so we do not lose this information. The threshold can be adjusted to make the profile as similar in length to the query as desired. In this manner, we can always overcome the expansion dilemma using a

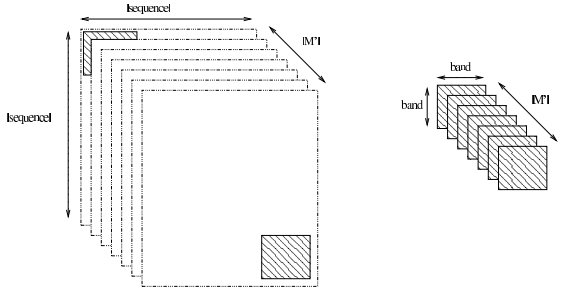

Fig. 3. A graphical representation for the space consumption addressed by the algorithms in Figure 2(b) and Figure 2(c). The boxed areas represent the allocated memory sections for the algorithm. Using PMFastR, the white space is not allocated, but virtualized by the functions mentioned previously. Any reference to the stripped area remains the same.

combination of an adjustment of the banding width and the quality threshold. We see this calculation in Equation (1) where k is the number of sequences and j is the column in question. Here $'-'$ represents the gap or delete character.

$$quality\_score(j) = \frac{\sum_{i=1}^{k}(c_{ij} \neq' -')}{k} \quad (1)$$

2.3 Multithread Design

The tree structure representation allows for another key observation: two sibling nodes are independent in calculation. We can see that while each node depends on the fact that the child node calculations be completed, the siblings (if more than one) do not depend on each other to fill in their own dynamic programming table. Using this fact, along with the widespread availability of multi-core machines, we can greatly improve the wall time of the algorithm. By running each branch of the tree on a separate thread, we can utilize all of the processing power simultaneously.

Only a few simple changes are needed to implement the multithreaded design. The major change is the separation of the iterative loop of recursing the tree into a function that can be called on any node. This is the basis of any thread that is spawned from the original instantiation. Within this new function, we must first make sure the calculations for the children are completed. This is where the recursion comes in, if there are two children of the current node, run them independently then wait for them both to complete before doing the parent node computation. Thus, the process begins on the root and recursively spawns threads for its children in a depth-first manner. Conversely, because all of the threads are paused until their children are complete the computation is done in a reverse breath-first manner; meaning the leafs are computed first, then the parent of the leafs and so on.

The running time of the algorithm (theoretically) is improved w-fold (where w is the maximum width of the tree). In most cases the program will be running on a single local machine and the thread handling will be done by the processor and operating system. However, we added one more feature that is needed for shared server environments. All of the computationally intensive code is done at the end of the thread call (the *Running Block*). If we are sharing a server in an uncontrolled environment, we may not want to have w threads all in the *Running Block* at once. We can then put this block for each node into a dispatch queue. When any thread is ready to enter *Running Block*, it is added to the queue. This queue is controlled by a dispatcher which releases the thread into this block, only letting a specific number of threads be in that state at any given time. This allows the user to control how many threads are running. While all of the threads will be spawned, the one that are waiting either on child threads or in the queue will be idle and not consuming resources.

2.4 Multiple Alignment

We expand the procedure to create a multiple alignment that is more accurate than those given by sequence only alignment tools. Additionally, we know that experimentally determining the structure of RNA in a lab environment is expensive and time consuming. Thus, we assume that we will only have a small number of structured sequences. In particular, we build a multiple alignment using structure information for only one sequence. We implemented a single pass progressive multiple alignment using the procedure presented above at the core. Each sequence in the input set was examined only once and then appended to a growing intermediate alignment.

The input is one sequence with experimentally determined structure and a database of sequences that are known to be in the same family but do not have attached annotated structure. After processing each new sequence chosen from the database we will obtain intermediate profile which contains both the structure and sequence information for the alignment, but may have large high-gap regions. This profile is then used as the input to the next step. Thus, the first step is to create the first profile by selecting one sequence from the database and aligning it to the input structured sequence. Once all sequences have been aligned, we then still only have an intermediate profile, though it now contains all of the sequences from the database. We then run a refinement step to clean the profile for output and analysis by other programs.

```
procedure processNode(v)
(* v is the current node to be processed. *)
if  v has one or more children
      spawn thread on processNode(left_child(v))
end if
if  v has two children
      spawn thread on processNode(right_child(v))
end if
wait for all (if any) child threads to complete
signal that this node is ready for processing
wait until resources are freed to process
execute PAln(v)
signal that the next node can be processed
```

Fig. 4. The multithreaded design elements of the improved algorithm. The number of threads spawned is the same as the number of nodes in the guide tree, but because of hierarchy, there is a limit on the number of running processes equal to the width of the tree.

Note here that there is a distinction between the single sequence-structure and the sequence-profile alignment procedures. While the underlying method remains exactly the same, we introduce a new element in the profile alignment, the Position Specific Scoring Matrix (PSSM), which is capable of storing the alignment column scoring parameters. Variation within a column of a profile can provide a lot of information when doing an alignment, thus aligning to a consensus sequence would defeat the purpose of building the profile to begin with. The PSSM is a manipulation of the scoring function that is used in the single sequence alignment. We are not simply checking two bases, we are comparing a base to a column. For each column, we have a probability distribution based on the contents of the column. We then assign a score according to this distribution. The initial score is based on the RIBOSUM nucleotide scoring matrix (NSM) as provided by Klein and Eddy [17] as used in PFastR [1]. Each entry in the scoring matrix for a node j and paired bases a and b is calculated as follows:

$$PSSM[j][a][b] = \sum_{c,d} \frac{n_{cd} * NSM[a][b][c][d]}{k} \qquad (2)$$

Here, a and b are the bases in the search sequence, c and d are all possible bases in the left and right columns of the profile, n_{cd} is the count of the number of columns in the profile that contains c in the left column and d in the right, and k is the total number of sequences. The NSM matrix provides a score for the replacement of base pair (a, b) with (c, d). Similarly, entries in the scoring matrix for a node representing an unpaired based at node j are calculated as:

$$PSSM[j][a] = \sum_{c} \frac{n_c * NSM[a][c]}{k} \qquad (3)$$

We see that this is only involving a single search base. Because the profile is constructed in a single pass, the order in which the sequences are added becomes very important. While this can be done in an ad-hoc manner and adequate results are achieved, we found that having some guidance greatly improved the output quality. Hence, we retrieve the alignment of the sequences in ClustalW [27] and use the ClustalW guide tree to direct the order of the input for PMFastR.

3 Results and Discussion

We evaluate the performance of PMFastR based on memory consumption, running time and quality of the alignments. First, we compare the memory consumption of PMFastR with that of its predecessor FastR. Then, we construct two 16s rRNA multiple alignments based on sequences from CRW and Greengenes. Later, we look at the improvement in the running time of the multi-threaded version of PMFastR. We also compare the performance of PMFastR to five other multiple alignment tools using a standardized dataset [12]. Finally, we generate multiple alignments for every RNA families from the Rfam 8.1 database [23] and compare them with the manually-curated seed alignments.

3.1 Memory Consumption

One of the goals of this project is to align large RNA sequences. In particular, we aim to align 16S rRNA and 23S rRNA using structural information. Given that the problem is $O(n^3)$ in both memory and time without banding, the first necessary development is to reduce the memory consumption to allow the algorithm to run on personal computers. Our new procedure theoretically reduces the memory consumption to $O(n)$. Here, we present empirical test to show that the memory consumption is indeed within those boundaries (see Figure 6). We tested our new procedure against FastR. The experiment consists of choosing a set of pairings of sequences from any given

procedure multipleAlignment
(* Here S is an array of sequence file names without extension. *)
run ClustalW to get the alignment tree for the non-structured sequences
$order$ the input sequences $\{S_1, S_2, S_3, ...S_k\}$ as ordered above
run sequence-structure alignment on S_0 with structure and S_1 without
$output$ the alignment to a profile file p.
for S_i as the rest of the sequences S_2 to S_k
    $compact$ p remove unpaired columns with less than cut_off sequences present
    run the profile-sequence alignment on p and S_i
    $reinsert$ the unpaired columns removed above
    $output$ this alignment to p
end for
$execute$ CMBuild –refine on p and $output$ the multiple alignment with structure annotation. *(optional)*

Fig. 5. The multiple alignment procedure. The progressive alignment procedure only examines one sequence from the input set at a time. As sequences are aligned, they are appended to the input profile and this becomes the input to the next iteration of the single alignment procedure. The process is concluded when the profile contains all of the sequences input.

set, then aligning them using the procedure presented here and with FastR. While our goal set is 16S rRNA, we could not run the benchmark algorithm on such a large sequence due to hardware limitations. Thus, we use a dataset comprising of tRNA, 5S rRNA, and cobalamine since both algorithms could run on this data. Results of these datesets are shown in Figure 6. The trend is clearly seen in this figure. The memory consumption was normalized by the length of the profile (n). We can see that the trend follows.

3.2 16S rRNA Alignment

As a proof of concept that PMFastR can be used to create quality alignments for long RNA sequences, we generated a large multiple alignment of 16S rRNA. Specifically, we constructed an alignment of the 567 16S rRNA from CRW database [28]. The average pairwise sequence identity was 48%. The alignment took $\sim$40 hours with a single process and the average maximum memory consumption per alignment of 3.63 Gb. This is the first multiple alignment of 16S RNA sequences which takes into account secondary structure information. This alignment is available on our supplementary data website.

As an additional test, we reassembled the Green-Genes core set [29], [30], which comprises of 4290 16S rRNA sequences. We used the *E. Coli* sequences with structure as the alignment seed. We then visualized the new alignment using both stack coloring and base coloring. The new alignment, as well as the visualizations, are available in the supplemental data on our website.

3.3 Multithreading

Because of the high constant on the running-time, we made algorithmic improvements by splitting the computation of sibling nodes on the tree into separate threads. The theoretical running time of the algorithm remains $O(nb^2)$; however, parallelization of a good portion of the algorithm allowed us to greatly reduce the wall time. By allowing just two concurrent threads, we can improve the running time by 78% for a subset of the CRW 16S rRNA dataset. The running time is further reduced when more concurrent threads are allowed. A ceiling is reached when the number of allowed processes is equal to the maximum width of the tree. Thus, when the structure of the RNA is more complex, we can see more improvements on running time by increasing the number of processes.

We can see that this is true by looking at the result in Table 1. For this set, we do not see much improvement when allowing more than 8 processes. It is important to note that this is the observed boundary and that is can be influenced by hardware. Specifically, the nodes in our cluster only had 8 processors, the memory latency could have had an impact on the running time boundary.

3.4 BRAliBase Data Sets

In order to assess the performance of our program, we show comparative results with other programs. We use the benchmark set BRAliBase 2.1 [31], which is based on seed alignments from the Rfam 7.0 database. We compare our program to four other structure-based RNA alignment tools (LARA, FOLDALIGNM [32], MARNA, and STRAL) and one purely sequence-based multiple alignment tool (MAFFT [33]). The results for LARA, FOLDALIGNM, MAFFT, MARNA, and STRAL were obtained from the supplementary data of Bauer *et al.* [12]. The full benchmarking results are available on the supplementary website.

In order to compare the alignments generated by PMFastR with the alignments generated by these other tools, we use four measures: Compalign, Sum-

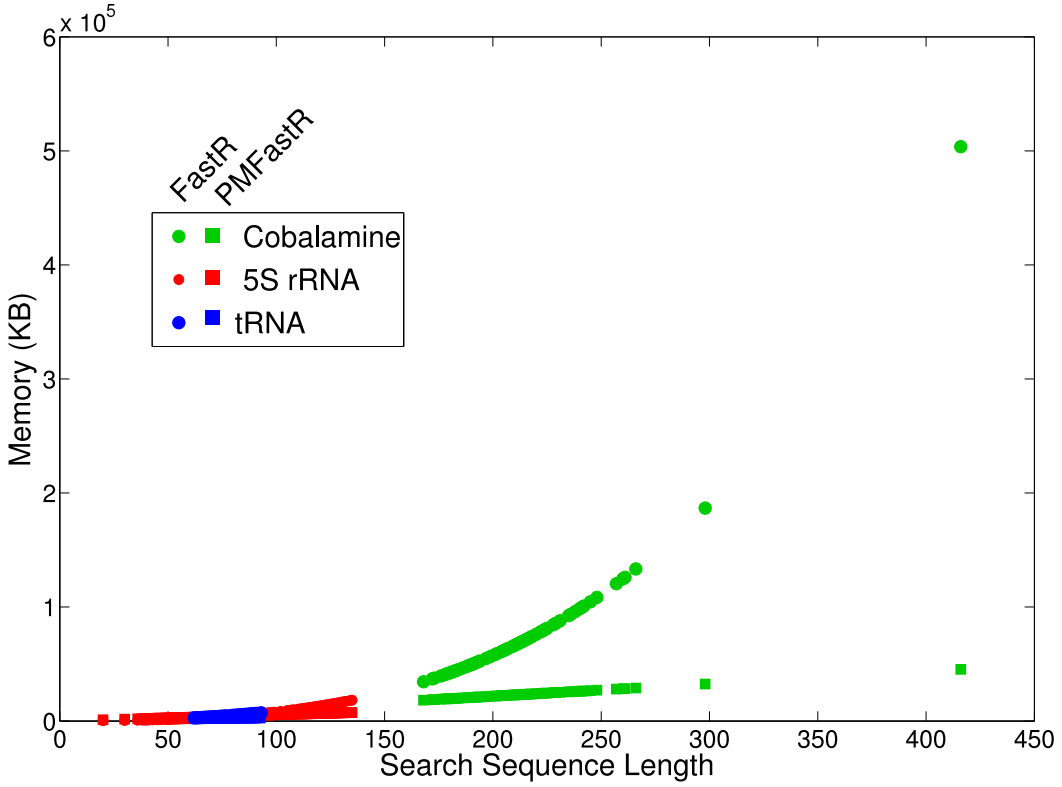

Fig. 6. Memory consumption for aligning the tRNA (blue), 5S rRNA (red) and cobalamine (green) sequences. The x-axis is the length of the search sequence, and the y-axis shows the average memory consumption for all tests with that length. The values were normalized using the profile length to show a cleaner representation of the information. The top line (circles) shows the data for the FastR algorithm, while the bottom line (squares) shows the PMFastR result.

TABLE 1
Multithreaded restriction results

Number of Processes	Total Time	Average Time Per Alignment	Speedup
1	7:10:46	4:18	1
2	4:10:38	2:25	1.78
4	3:24:00	2:02	2.11
8	2:59:13	1:48	2.40
16	2:57:14	1:46	2.43
32	2:56:51	1:46	2.44

of-Pairs Score (SPS), Structure Conservation Index (SCI) and Structure Conservation Rate (SCR). Sum-of-Pairs Score (SPS) is the fraction of pairs aligned in both the reference alignments and the predicted alignments and has been used in many alignment benchmarks [24], [34]. It is similar to Compalign which has been used in LARA's benchmarking [12]. For SPS and Compalign, a value of 1 represents the reference is the same as the test alignment. On the other hand, Structure Conservation Rate (SCR) calculates the fraction of the conserved base pairs in the alignments. It resembles the Structure Conservation Index (SCI) [24], [4] with the differences that SCR rewards the compensatory mutations and uses reference structure from the benchmarking alignment while SCI first uses RNAalifold [35] to predict the consensus structures and then calculates the structural conservation index with compensatory mutations. Since we already have the structural information of the benchmarking data set, SCR is a better way to check the paired regions alignments. Since other programs (LARA, FOLDALIGNM, MARNA, and STRAL) do not output the structural information, we have to use RNAalifold to predict the consensus structure for SCR.

The BRAliBase test set is divided into six groups.

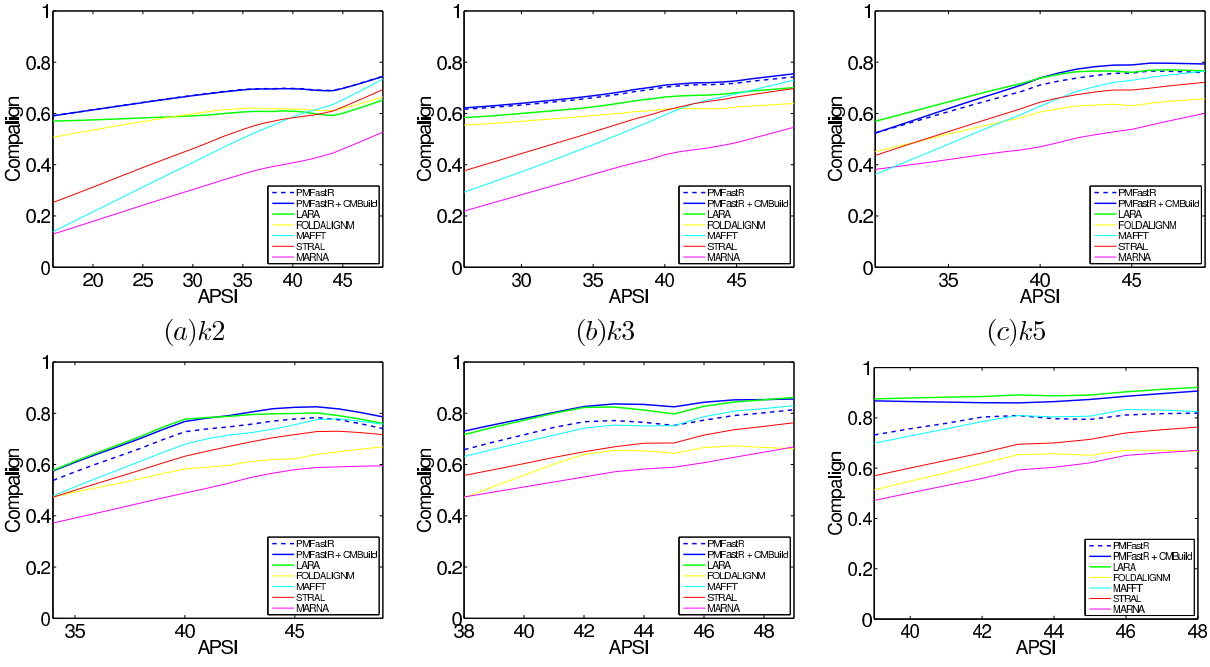

Fig. 7. Compalign benchmarking. Each alignment was run though Compalign, this score is shown on the y-axis. This is a measure of the similarity of an input alignment with the reference alignment. The x-axis shows the average pairwise sequence identity (APSI) for each dataset. The data used were sets of (a)2, (b)3, (c)5, (d)7, (e)10, and (f)15 sequences per dataset.

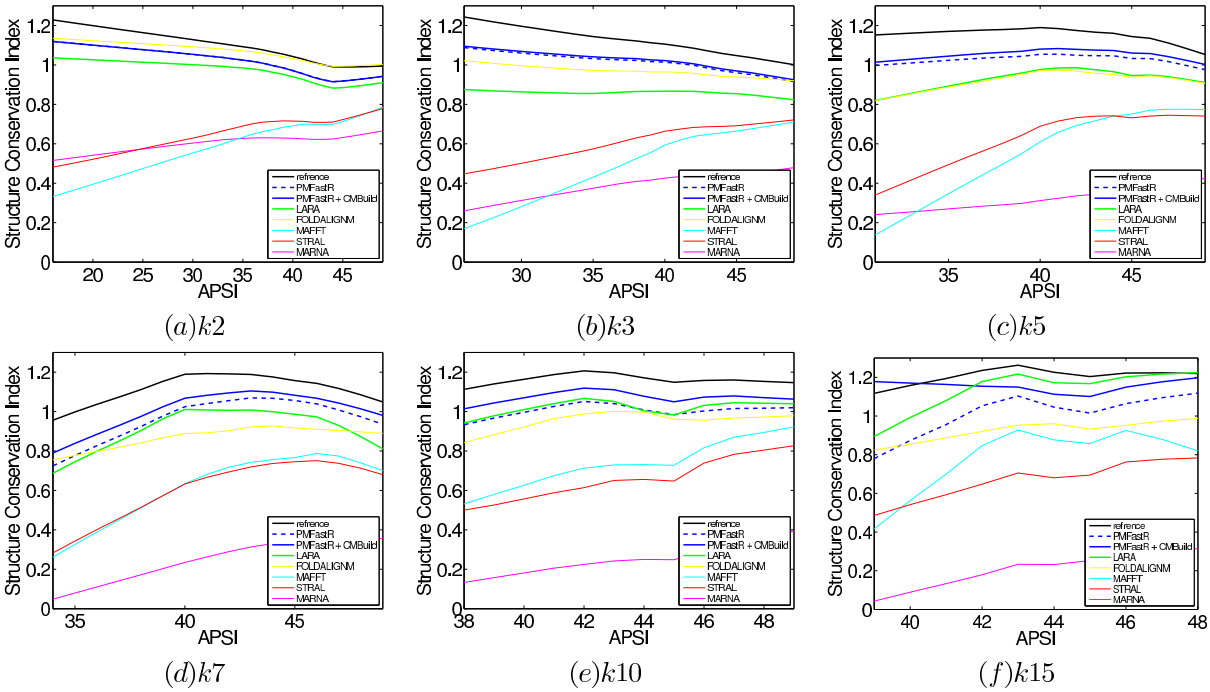

Fig. 8. Structure Conservation Index (SCI) benchmarking. SCI score is obtained for each alignment, which is shown on the y-axis. This is a measure of the rate at which the pairs that are annotated as structural follow the canonical bases using RNAlifold. The x-axis shows the average pairwise sequence identity (APSI) for each dataset. The data used were sets of (a)2, (b)3, (c)5, (d)7, (e)10, and (f)15 sequences per dataset.

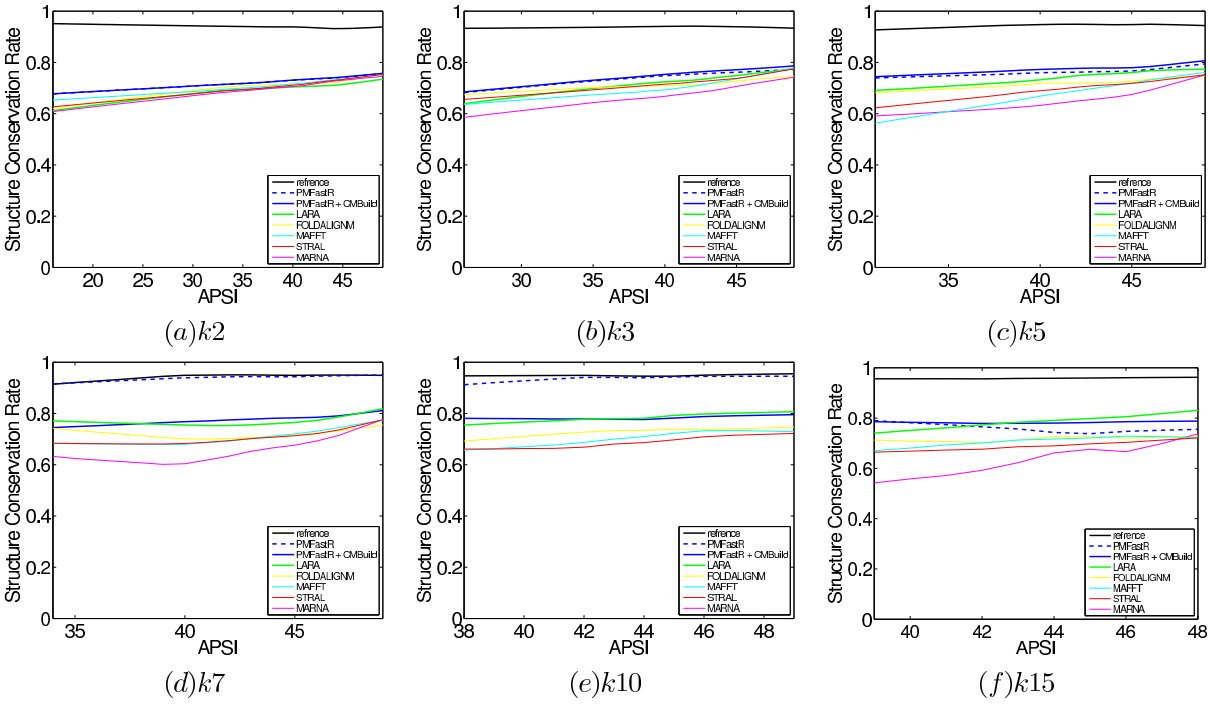

Fig. 9. Structure Conservation Rate (SCR) benchmarking. SCR score is obtained for each alignment, which is shown on the y-axis. This is a measure of the rate at which the pairs that are annotated as structural follow the canonical bases using the algorithmic structure (when available). The x-axis shows the average pairwise sequence identity (APSI) for each dataset. The data used were sets of (a)2, (b)3, (c)5, (d)7, (e)10, and (f)15 sequences per dataset.

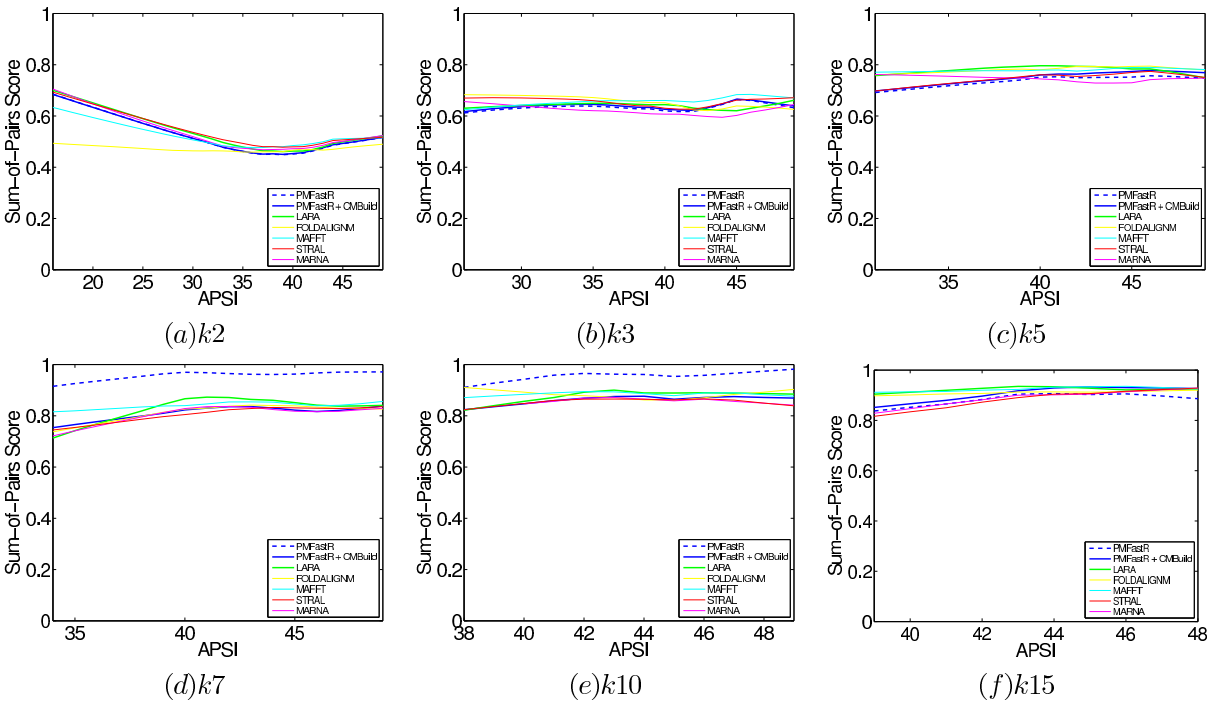

Fig. 10. Sum of Pairs Score (SPS) benchmarking. SPS is obtained for each alignment, which is shown on the y-axis. This is a measure of the alignments similarity to the goal alignment. The x-axis shows the average pairwise sequence identity (APSI) for each dataset. The data used were sets of (a)2, (b)3, (c)5, (d)7, (e)10, and (f)15 sequences per dataset.

Each group contains a number of sets of sequences from the same family. These six groups have a different number of sequences per set, ranging form 2 to 15. Within each set the sequence identity is also identified. We choose sequence such as to have a variation in sequence identity ranging from 39% to 50%. We ignore test sets with sequence identity above 50% since we want to test in the range where structural information is integral to the RNA alignment. Results are shown in Figures 7, 8, 9 and 10. Additionally, we show the results for other multiple alignment programs on the same sets. We can see that in all of the tests we performed, PMFastR is comparable or better than LARA and the other tested programs. In the few test cases where LARA does have a better score on the high sequence identity sets, PMFastR has a better score at the low end, which is where our study was focused.

3.5 Rfam Families

Rfam is a well-known database of families of RNA sequences. The data comes from multiple locations and in most cases the alignments are hand-curated or seeded with a hand-curated alignment. The repository consists of 607 families of RNA, each with a seed and full alignment. We use version 8.1 as a benchmark alignment. We retrieve all families from the database and remove one sequence from the family that was most consistent with the structure. Each sequence in the multiple alignment is then separated from the group with gaps removed, then fed into PMFastR individually without the annotated structure. We then obtain a new multiple alignment from the same data set as in the benchmark. We can then examine the alignment to assess its quality. We use the same analysis tools mentioned in the BRAliBase comparisons. It can be seen from this analysis that PMFastR is able to regenerate very similar alignment to those created by hand. The details of these comparisons and the entire regenerated Rfam 8.1 database are available in the supplemental data.

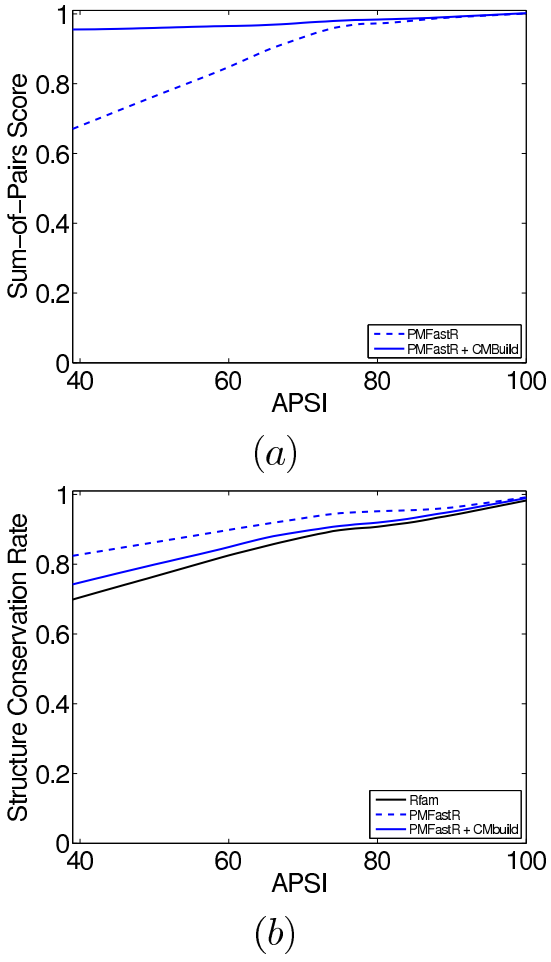

Fig. 11. Benchmarking of Rfam seed alignments. Comparison of alignments generated by PMFastR on RFam families with manually-curated seed alignments by two measures, SPS (a) and SCR (b). Each family was stripped of the structure except one sequence which was used as the seed, then reconstructed using PMFastR.

4 Conclusions

We presented an algorithm, PMFastR, which aligns RNA sequences using both sequence and structure information. The algorithm only requires one sequence to have structure information and is able to align all other input sequences without human interaction. Because we are able to drastically reduce the memory consumption as compared to previous work, our algorithm is able to run on very long RNA sequences, such as 16S and 23S rRNA.

We propose three major ideas which improve the performance of PMFastR and which can be applied to other work. Banding allows a significant reduction in both run time and space consumption of our alignment algorithm. In fact, we are able to reduce the space consumption from cubic to linear when comparing to the predecessor, FastR. Moreover, reordering the inner loops of this algorithm allows us to run it in a multi-threaded manner, thus drastically reducing the wall time. These modifications do not alter the quality of the algorithm's results. In fact, PMFastR with CMBuild refinement performs comparably to other state-of-the-art RNA multiple alignment tools and creates comparable alignments to the hand-curated ones in the Rfam database. All results, as well as the application source code, can be found on the project web page at http://genome.ucf.edu/PMFastR. While PMFastR performs well for the examples shown in this paper, there seems to be an implicit limit on the number of sequences that can be alignment at a high quality. This

is most likely due to the single pass progressive nature of the program.

ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation grant NSF-DBI:0516440 and the UCF STOKES High Performance Computing Cluster project.

REFERENCES

[1] S. Zhang, I. Borovok, Y. Aharonowitz, R. Sharan, and V. Bafna, "A sequence-based filtering method for ncrna identification and its application to searching for riboswitch elements," *Bioinformatics*, vol. 22, no. 14, pp. e557–e565, 2006.

[2] S. Eddy and R. Durbin, "Rna sequence analysis using covariance models," *NAR*, vol. 22, pp. 2079–2088, 1994.

[3] E. Rivas and S. Eddy, "Noncoding RNA gene detection using comparative sequence analysis," *BMC Bioinformatics*, vol. 2, pp. 8–26, 2001.

[4] S. Washietl, I. L. Hofacker, M. Lukasser, A. Hottenhofer, and P. F. Stadler, "Mapping of conserved RNA secondary structures predicts thousands of functional noncoding RNAs in the human genome," *Nat. Biotechnol.*, vol. 23, pp. 1383–1390, Nov 2005.

[5] C. B. Do, M. S. Mahabhashyam, M. Brudno, and S. Batzoglou, "Probcons: Probabilistic consistency-based multiple sequence alignment." *Genome Res.*, vol. 15, no. 2, pp. 330–340, February 2005.

[6] R. Edgar, "MUSCLE: multiple sequence alignment with high accuracy and high throughput," *Nucleic Acids Res.*, vol. 32, pp. 1792–1797, 2004.

[7] C. Notredame, D. G. Higgins, and J. Heringa, "T-coffee: A novel method for fast and accurate multiple sequence alignment." *J Mol Biol*, vol. 302, no. 1, pp. 205–217, September 2000.

[8] J. D. Thompson, D. G. Higgins, and T. J. Gibson, "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucl. Acids Res.*, vol. 22, no. 22, pp. 4673–4680, 1994.

[9] J. A. Jaeger, D. H. Turner, and M. Zuker, "Improved predictions of secondary structures for RNA," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 86, no. 20, pp. 7706–7710, 1989.

[10] B. Knudsen and J. Hein, "Pfold: RNA secondary structure prediction using stochastic context-free grammars," *Nucl. Acids Res.*, vol. 31, no. 13, pp. 3423–3428, 2003.

[11] M. Zuker and D. Sankoff, "RNA secondary structures and their prediction," *Bulletin of Mathematical Biology*, vol. 46, no. 4, pp. 591–621, 1984.

[12] M. Bauer, G. Klau, and K. Reinert, "Accurate multiple sequence-structure alignment of rna sequences using combinatorial optimization," *BMC Bioinformatics*, vol. 8, no. 1, p. 271, 2007.

[13] T. Jiang, G. Lin, B. Ma, and K. Zhang, "A general edit distance between rna structures." *Journal of Computational Biology*, vol. 9, no. 2, pp. 371–388, 2002.

[14] D. Sankoff, "Simulations solution of the RNA folding, alignment and protosequence problems," *SIAM J. Appl. Math.*, vol. 45, no. 5, pp. 810–825, 1985.

[15] S. Zhang, B. Haas, E. Eskin, and V. Bafna, "Searching genomes for noncoding rna using fastr," *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, vol. 2, no. 4, pp. 366–379, Oct.-Dec. 2005.

[16] Z. Weinberg and W. L. Ruzzo, "Exploiting conserved structure for faster annotation of non-coding RNAs without loss of accuracy," *Bioinformatics*, vol. 20, no. suppl_1, pp. i334–341, 2004.

[17] R. Klein and S. Eddy, "Rsearch: Finding homologs of single structured rna sequences," *BMC Bioinformatics*, vol. 4, no. 1, p. 44, 2003.

[18] I. L. Hofacker, S. H. Bernhart, and P. F. Stadler, "Alignment of RNA base pairing probability matrices," *Bioinformatics*, vol. 20, pp. 2222–2227, Sep 2004.

[19] S. Siebert and R. Backofen, "MARNA: multiple alignment and consensus structure prediction of RNAs based on sequence structure comparisons," *Bioinformatics*, vol. 21, pp. 3352–3359, Aug 2005.

[20] I. Holmes, "Accelerated probabilistic inference of RNA structure evolution," *BMC Bioinformatics*, vol. 6, p. 73, 2005.

[21] D. Dalli, A. Wilm, I. Mainz, and G. Steger, "STRAL: progressive alignment of non-coding RNA using base pairing probability vectors in quadratic time," *Bioinformatics*, vol. 22, pp. 1593–1599, Jul 2006.

[22] M. Hochsmann, B. Voss, and R. Giegerich, "Pure multiple RNA secondary structure alignments: a progressive profile approach," *IEEE/ACM Trans Comput Biol Bioinform*, vol. 1, pp. 53–62, 2004.

[23] S. Griffiths-Jones, S. Moxon, M. Marshall, A. Khanna, S. R. Eddy, and A. Bateman, "Rfam: annotating non-coding RNAs in complete genomes," *Nucl. Acids Res.*, vol. 33, no. suppl_1, pp. D121–124, 2005.

[24] P. P. Gardner, A. Wilm, and S. Washietl, "A benchmark of multiple sequence alignment programs upon structural RNAs," *Nucleic Acids Res.*, vol. 33, pp. 2433–2439, 2005.

[25] V. Bafna, S. Muthukrishnan, and R. Ravi, "Computing similarity between RNA strings," in *Combinatorial Pattern Matching*, ser. Lecture Notes in Computer Science, Z. Galil and E. Ukkonen, Eds. Springer Berlin / Heidelberg, 1995, vol. 937, pp. 1–16.

[26] V. Bafna and S. Zhang, "FastR: fast database search tool for non-coding RNA," *Proc IEEE Comput Syst Bioinform Conf*, pp. 52–61, 2004.

[27] M. Larkin, G. Blackshields, N. Brown, R. Chenna, P. McGettigan, H. McWilliam, F. Valentin, I. Wallace, A. Wilm, R. Lopez, J. Thompson, T. Gibson, and D. Higgins, "Clustal W and Clustal X version 2.0," *Bioinformatics*, vol. 23, no. 21, pp. 2947–2948, 2007.

[28] J. Cannone, S. Subramanian, M. Schnare, J. Collett, L. D'Souza, Y. Du, B. Feng, N. Lin, L. Madabusi, K. Muller, N. Pande, Z. Shang, N. Yu, and R. Gutell, "The comparative rna web (crw) site: an online database of comparative sequence and structure information for ribosomal, intron, and other rnas," *BMC Bioinformatics*, vol. 3, no. 1, p. 2, 2002.

[29] T. Z. DeSantis, I. Dubosarskiy, S. R. Murray, and G. L. Andersen, "Comprehensive aligned sequence construction for automated design of effective probes (CASCADE-P) using 16S rDNA," *Bioinformatics*, vol. 19, pp. 1461–1468, Aug 2003.

[30] T. Z. DeSantis, P. Hugenholtz, N. Larsen, M. Rojas, E. L. Brodie, K. Keller, T. Huber, D. Dalevi, P. Hu, and G. L. Andersen, "Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB," *Appl. Environ. Microbiol.*, vol. 72, pp. 5069–5072, Jul 2006.

[31] A. Wilm, I. Mainz, and G. Steger, "An enhanced RNA alignment benchmark for sequence alignment programs," *Algorithms Mol Biol*, vol. 1, p. 19, 2006.

[32] E. Torarinsson, J. H. Havgaard, and J. Gorodkin, "Multiple structural alignment and clustering of RNA sequences," *Bioinformatics*, vol. 23, pp. 926–932, Apr 2007.

[33] K. Katoh, K. Kuma, H. Toh, and T. Miyata, "MAFFT version 5: improvement in accuracy of multiple sequence alignment," *Nucleic Acids Res.*, vol. 33, pp. 511–518, 2005.

[34] J. D. Thompson, F. Plewniak, and O. Poch, "A comprehensive comparison of multiple sequence alignment programs," *Nucleic Acids Res.*, vol. 27, pp. 2682–2690, Jul 1999.

[35] I. L. Hofacker, M. Fekete, and P. F. Stadler, "Secondary structure prediction for aligned RNA sequences," *J. Mol. Biol.*, vol. 319, pp. 1059–1066, Jun 2002.

Dan DeBlasio received a BS and MS in Computer Science from the University of Central Florida in 2007 and 2009 respectively. He is currently a Ph.D. student in Computer Science at the University of Arizona in Tucson and is a fellow of the IGERT in Genomics. He is also a student member of IEEE and the IEEE Computer Society.

Jocelyne Bruand is a Ph.D. student in the Bioinformatics and Systems Biology Graduate Program at the University of California, San Diego under Dr. Bafna and Dr. Macagno. She received her B.S. in Information and Computer Science from the University of California, Irvine in 2005.

Shaojie Zhang received his B.S. in Computer Science from Peking University, Beijing, China, M.Eng. from Nanyang Technological University, Singapore, and Ph.D. in Computer Science from the University of California, San Diego. He is currently an Assistant Professor in the Department of Electrical Engineering and Computer Science at the University of Central Florida. His research is focused on bioinformatics, which includes ncRNA gene finding, RNA analysis, and computational genomics.