# PMFastR: A New Approach to Multiple RNA Structure Alignment

Daniel DeBlasio[1], Jocelyne Bruand[2], Shaojie Zhang[1]

[1] University of Central Florida, Orlando, FL. 32816, USA
{deblasio, shzhang}@eecs.ucf.edu
[2] University of California, San Diego, La Jolla, CA 92093, USA
jbruand@ucsd.edu

**Abstract.** Multiple RNA structure alignment is particularly challenging because covarying mutations make sequence information alone insufficient. Many existing tools for multiple RNA alignments first generate pairwise RNA structure alignments and then build the multiple alignment using only the sequence information. Here we present PMFastR, an algorithm which iteratively uses a sequence-structure alignment procedure to build a multiple RNA structure alignment. PMFastR 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. Finally, we present results on benchmark data sets from BRAliBase, which shows PMFastR outperforms other state-of-the-art programs. Furthermore, we regenerate 607 Rfam seed alignments and show that our automated process creates similar multiple alignments to the manually-curated Rfam seed alignments.

**Key words:** multiple RNA alignment, RNA sequence-structure alignment, iterative alignment

## 1 Introduction

A high quality multiple alignment of RNA sequences is crucial for RNA homology search [**?**], structure analysis [**?**] and discovery [**?**,**?**]. Even though methods for multiple alignments of DNA and protein sequences are well studied [**?**,**?**,**?**,**?**], multiple RNA structure alignment is still an open problem.

The problem of aligning two RNA sequences with consideration of the structural information comes as an extension of the RNA structure prediction studies [**?**,**?**,**?**]. When aligning two RNA sequences, we can consider three problems and their associated methods depending on the availability of the structural information for these RNA sequences [**?**,**?**]. 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 [**?**]; this is the RNA *sequence-sequence* alignment problem. Another potential input is a structural profile or other structural information for one of the sequences but not for the second one; this is the *sequence-structure* alignment problem. Several methods have been developed for this problem and are used for finding RNA homologs. FastR [**?**] and PFastR [**?**]

use a guided trees and dynamic programming to a globally align a sequence or profile to a given target sequence. CMSEARCH [**?**] and RSEARCH [**?**] 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 [**?**]; this is the RNA *structure-structure* alignment problem.

Much work has already been done on the multiple RNA structure alignment problem. Most of these RNA multiple alignment methods (PMmulti [**?**], MARNA [**?**], Stemloc [**?**], STRAL [**?**], and LARA [**?**]) use pairwise RNA alignment (sequence-sequence or structure-structure) for all sequences and then combine these alignments into a multiple alignment using T-Coffee [**?**] or other progressive strategies. For instance, LARA can take in the structure or predict the structure of the input sequences. The program then uses a connection graph and integer linear programming to create pairwise alignments. The score of these pairwise alignments are then fed into T-Coffee to generate a multiple alignment. 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. On the other hand, structure-structure alignment is not feasible on very long RNA sequences, such as 16S rRNA and 23S rRNA. Here, we use the RNA sequence-structure alignment strategy to build a multiple alignment. Eddy and Durban [**?**] have used this strategy in the past by using a covariance model but their algorithm requires an initial multiple alignment as input. In contrast, the algorithm presented here requires only one sequence with structure and a database of other unaligned sequences as input.

Databases of multiple RNA alignments such as Rfam [**?**] maintain very high quality alignments which are obtained by integration of multiple sources and manual curation. We use these databases for baseline comparison as a mean to assay how well our algorithm is performing. We show in this paper that the proposed algorithm can produce comparable results without manually curation of the alignments.

In this paper, we present the Profile based Multiple Fast RNA Alignment (PMFastR) algorithm. An overview of this program is presented in Figure **??**. PMFastR does a multiple structure-based alignment from scratch while using a relatively small amount of memory and can be used to make a multiple structure alignment of long RNA sequences such as 16S rRNA or 23S rRNA sequences. The input is one sequence, with structure information, and a group of sequences without such information. Our algorithm consists of three major steps. The first step is a structure-sequence alignment of an RNA sequence from the database with the original structure. This second step outputs an aligned profile of two sequences with structure information. We can then align the next element from the database to the output profile and obtain a new alignment profile. This can be repeated iteratively until all of the elements of the input dataset are aligned. Finally, we run the CMBuild refinement program 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 sec-

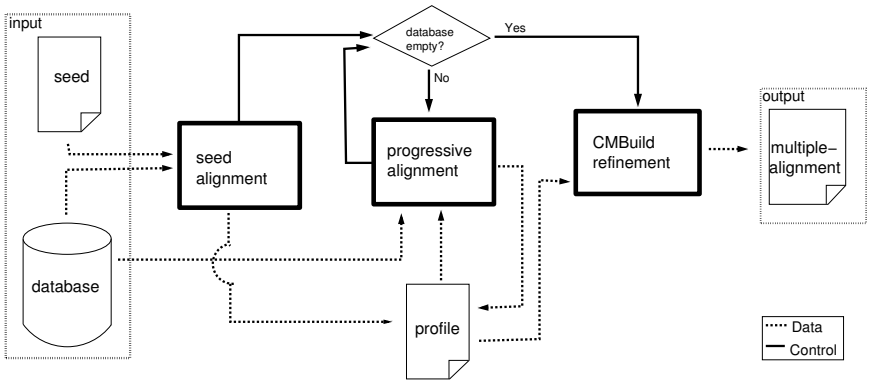



**Fig. 1.** Overview of PMFastR. The three major steps are highlighted in the solid square boxes, while the input and output of the program are represented in dashed boxes. After generation of a seed alignment, the algorithm progressively aligns the rest of the input data set. Finally, we run the CMBuild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

tion, we run several benchmarking tests on PMFastR with other multiple RNA alignment tools using BRAliBase (version 2.1) [**?**] as test data sets, which are extracted from Rfam database. We also regenerate all Rfam hand-curated seed alignments (version 8.1) [**?**] automatically with comparable results.

## 2 Methods

PMFastR is based on the methods used in FastR [**?**] and PFastR [**?**] alignment procedures, which were designed for RNA sequence-structure alignment. This section first briefly describes the sequence-structure alignment algorithm itself, then the improvements made upon it. These improvements are banding, multithreading, and multiple alignment. By using banding, the algorithm has a reduced running time and space consumption by several orders of magnitude. Multithreading allows the algorithm to utilize multiple cores when available to further improve wall time. Finally, by using a progressive profile alignment, the algorithm is able to produce multiple sequence alignments.

### 2.1 The Alignment Procedure

We make the assumption that all base-pairs are non-crossing, and let $M$ be the set of all such base-pairs. Thus, 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 $j' < j'' < j$. The alignment can be done by recursing on a tree which is representing the RNA profile with the secondary structural information. Since this tree can have high degree and not all columns of the profile participate in it, we binarize it using the procedure given in FastR [**?**]. The number of nodes in this tree is $O(m)$, where $m$ is the number of columns

in the profile. Figure **??**(a) describes a dynamic programming algorithm for aligning a sequence to an RNA profile. The RNA profile is then modeled as a tree as described above. Each node $v$ in the tree either corresponds to a base pair $(l_v, r_v) \in M$ of the profile, an unpaired base of the profile (and has its own PSSM), or a branching point in a pair of parallel loops. The alignment of the sequence to the RNA profile is done by recursing on the tree representing the input RNA profile.

### 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 matching. The memory 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 entails 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 $\sim O(n^2 * m)$ to $\sim O(band^2 * m)$, 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 **??** and **??** show the methods used for mapping into the new bounded array. We can see that these procedures incur only a small amount of overhead, thus the effect on running time is not significant.

A problem arises when we work with sequences that are not of similar length. To overcome this we can make adjustments to the analysis parameters to allow for these alignments to still be computed. The first solution is to adjust the banding parameter to search a larger space when the difference in size is high. In particular, the banding value should always be more than one and a half times the difference in length. Additionally, any columns in the profile that do not meet a certain quality criteria (percentage of sequences represented) are removed before alignment. They are added back when the profile is output, by adjusting the quality criteria we can also effect the length of the profile used for alignment.

**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 $band * band * m$.

```
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. 3.** The multithreaded design elements of the improved algorithm. Because the threads are queued, we can control the amount of simultaneous computation.

### 2.3 Multithread Design

To improve the feasibility on large sequences, parallelization is used to improve the wall time of PMFastR. The intuition comes from the fact that, at any given time, only one node is processed from the tree. Each node only depends on its two children, which in turn depend only on their children, and so on. This means that the two children do not depend on each other and can be processed simultaneously and independently.

By altering the procedure in Figure **??** to take the node $v$ as input, we can run each node independently. Another procedure is used to manage the threading: given a node, it runs the alignment on its two children (if any exist), then runs the alignment on the input node. If each of the children is spawned as a new thread, then this can be carried out by the processing environment in parallel.

This improvement allows the majority of the processing time to be run in parallel. Figure **??** shows the node processing procedure. A *signal* and *wait* queue is used to have processes wait for resources allocated. This allows a restriction on the amount of processor being used in a shared environment by giving control to the user of how many active process threads are allowed to be in the "running state" at any given point.

### 2.4 Multiple Alignment

We implemented a single pass progressive profile multiple alignment. The algorithm first aligns one input sequence with structure and the rest of the given sequences without structure. Note that we can also give a profile rather than a single sequence. The first step involves running the profile alignment algorithm described above on the input profile and a single sequence from the set, which outputs an alignment between the sequence and the profile. We then use this alignment to create a new profile composed of the original profile (with additional columns for the gaps inserted in the alignment) followed by the aligned (gapped) sequence.

Since 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 from ClustalW [**?**] and use the ClustalW guided tree to direct the order of the input for PMFastR.

Figure **??** shows the complete alignment procedure. This procedure assumes that the input is a single sequence with structure and a set of sequences without structure. Note the differences between the sequence-structure alignment and the profile-sequence alignment; a PSSM is used to help the alignment in the profile alignment, whereas this step is excluded from the sequence-structure alignment. If the input is a profile instead of a single sequence, the third and fourth lines of the algorithm are be skipped. Finally, we also use the refinement option of CMBuild [**?**] to refine the reinserted unpaired columns.

**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.

**Fig. 4.** The multiple alignment procedure. The profile is built progressively using the structure from a seed sequence. All subsequent sequences are then aligned to this profile.

## 3 Experimental Results

We evaluated the performance of PMFastR based on running time, quality of the alignments and memory consumption. We first looked at the improvement in the running time of the multi-threaded version of PMFastR in function of the number of simultaneous processes. We then compared the performance of PMFastR to five other multiple alignment tools [**?**]. We also generated multiple alignments for every RNA family from the Rfam 8.1 database [**?**] and evaluated their quality by comparing them to that of the manually-curated seed alignments. Furthermore, we assessed the improvements in memory consumption of the PMFastR in contrast to that of its predecessor FastR; these results are shown in the supplementary data.

### 3.1 Multithreading

Multithreading can be used to improve the running time of PMFastR. We tested the improvement in performance from multithreading by running our algorithm on 100 alignments on a multi-core system while varying the number of active threads. In Table 1, we show the total wall time needed to complete the jobs on four nodes of a high performance cluster with eight processors per node. The speedup was calculated as the original runtime over the improved runtime. It can be seen that for more than eight processes, the performance increase is minimal. This is due to the dominance of communication overhead as the load on each thread diminishes.

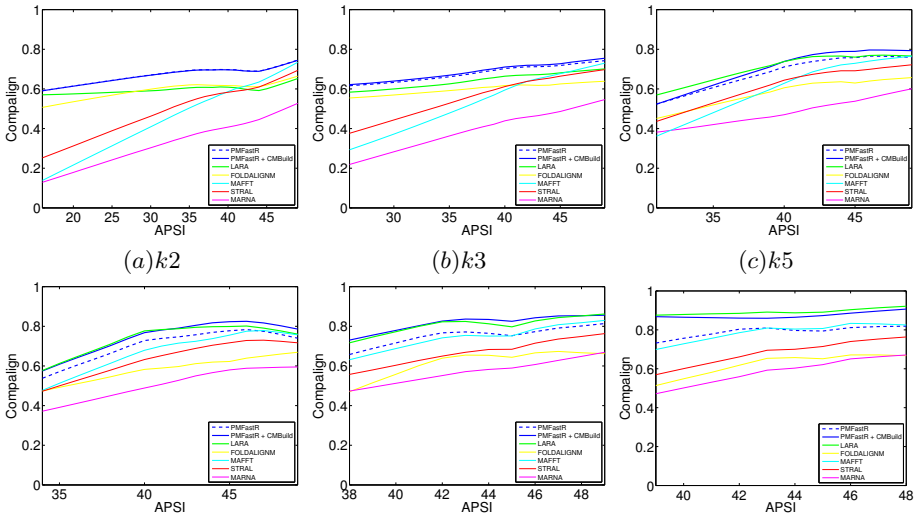
**Table 1.** Multithreaded restriction results

| 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 |

### 3.2 Alignments on BRAliBase Data Sets and Rfam Families

We chose the widely used benchmark set BRAliBase 2.1 [**?**], which is based on seed alignments from the Rfam 7.0 database, as our test set. We compared PMFasR to four other structure-based RNA alignment tools (LARA, FOLDALIGNM [**?**], MARNA, and STRAL) and one purely sequence-based multiple alignment tool (MAFFT [**?**]).

In order to compare the alignments generated by PMFastR with the alignments generated by these other tools, we used four measures: Compalign, Sum-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 [**?**,**?**]. It is similar to Compalign which has been used in LARA's benchmarking [**?**]. For SPS and Compalign, a value of 1 is achieved when the reference alignment 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) [**?**,**?**] with the differences that SCR rewards the compensatory mutations and uses reference structure from the benchmarking alignment, while SCI first uses RNAalifold [**?**] 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




$(a)k2$ $(b)k3$ $(c)k5$

$(d)k7$ $(e)k10$ $(f)k15$

**Fig. 5.** Compalign Benchmarking. Each alignment was run though Compalign, this score is shown on the y-axis. 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.

(LARA, FOLDALIGNM, MARNA, and STRAL) do not output the structural information, we have to use RNAalifold to predict the consensus structure for SCR.

To remain consistent with Bauer *et al.* [**?**], Figure **??** and Figure **??** show Compalign and SCI results for benchmarking data set. We can see that PMFastR with CMBuild refinement outperforms the other programs in these tests. The results for LARA, FOLDALIGNM, MAFFT, MARNA, and STRAL were obtained from the supplementary data of Bauer *et al.* [**?**]. Due to space restrictions, the full benchmarking results, including the SPS and SCR test results, are available on the supplementary website.

To further test our program, we use PMFastR with CMBuild to recreate all seed alignments from Rfam 8.1 database starting from only one sequence with its annotated structure. There are 607 families in the Rfam database. The detailed alignment results of all RFam families are available in the supplementary data. Figure **??** shows that alignments predicted by PMFastR with CMBuild are comparable with the manually curated seed alignments from the Rfam database. Moreover, we generated a multiple structure alignment for the 567 16S rRNA from CRW database [**?**] with average pairwise sequence identity of 48%. The alignment took ∼40 hours, and had an average maximum memory consumption per alignment of 3.63 Gb. To our knowledge, this is the first automated multiple alignment of 16S RNA sequences which takes into account secondary structure information. This alignment is available on our supplementary data website.

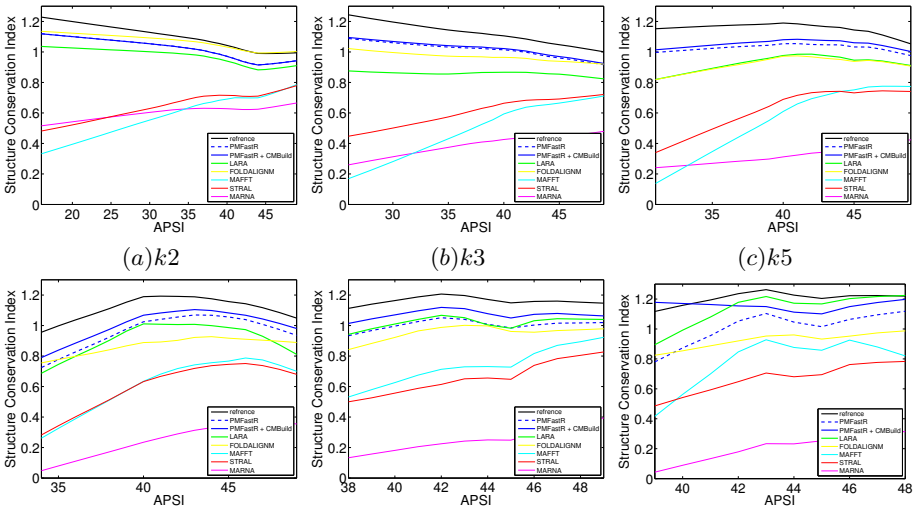



**Fig. 6.** SCI Benchmarking. Structure Conservation Index (SCI) score is obtained for each alignment, which is shown on the y-axis. The x-axis shows the average pairwise sequence identity (APSI) for each dataset. The data used were sets of (a)2, (b)3, (c)5, (d)7, (e)10, and (f)15 sequences per dataset.

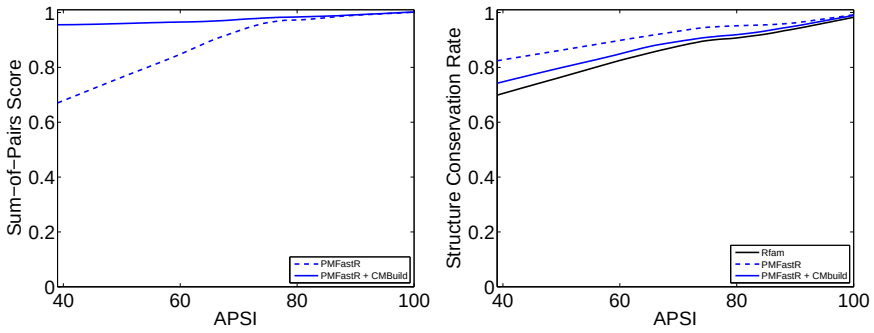
## 4 Conclusion

We presented an algorithm 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 improved 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, and we show that PMFastR with CMBuild refinement does better than 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.

## Acknowledgements

Many of the results shown where produced using the UCF STOKES High Performance Computing cluster.




(a) (b)

**Fig. 7.** Comparison of alignments generated by PMFastR on RFam families with manually curated seed alignments by two measures, SPS (a) and SCR (b).