# New Computational Approaches For Multiple Rna Alignment And Rna Search

2009

Daniel DeBlasio
*University of Central Florida*

# New Computational Approaches For Multiple RNA Alignment And RNA Search

by

Daniel F. DeBlasio
B.S. University of Central Florida, 2007

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2009

Major Professor:
Shaojie Zhang

# Abstract

In this thesis we explore the the theory and history behind RNA alignment. Normal sequence alignments as studied by computer scientists can be completed in $O(n^2)$ time in the naive case. The process involves taking two input sequences and finding the list of edits that can transform one sequence into the other. This process is applied to biology in many forms, such as the creation of multiple alignments and the search of genomic sequences. When you take into account the RNA sequence structure the problem becomes even harder.

Multiple RNA structure alignment is particularly challenging because covarying mutations make sequence information alone insufficient. Existing tools for multiple RNA alignments first generate pair-wise 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 also has low memory consumption allowing for the alignment of large sequences such as 16S and 23S rRNA. Specifically, we reduce the memory consumption to $\sim O(band^2 * m)$ where $band$ is the banding size. Other solutions are $\sim O(n^2 * m)$ where $n$ and $m$ are the lengths of the target and query respectively. The algorithm also provides a method to utilize a multi-core environment. 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.

While these methods can also be applied directly to genome sequence search, the abundance of new multiple species genome alignments presents a new area for exploration. Many multiple alignments of whole genomes are available and these alignments keep growing in size. These alignments can provide more information to the searcher than just a single sequence. Using the methodology from sequence-structure alignment we developed AlnAlign, which searches an entire genome alignment using RNA sequence structure. While programs have been readily available to align alignments, this is the first to our knowledge that is specifically designed for RNA sequences. This algorithm is presented only in theory and is yet to be tested.

# Acknowledgments

The author would like to thank his advisor and committee chair Dr. Shaojie Zhang for his support and help though this process and his input on the ideas presented. They would also like to thank Dr. Hu and Dr. Li for their input and comments. Finally, they would like to thank Jocelyn Braund for her contributions to the PMFastR paper. Chapter 2, in part, is a reprint of the paper, "PMFastR: A New Approach to Multiple RNA Structure Alignment", co-authored with Jocelyn Braund and Shaojie Zhang in *The 9th Workshop on Algorithms in Bioinformatics (WABI)*.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1:
# INTRODUCTION

## 1.1 GENERAL BACKGROUND

### 1.1.1 String Algorithms

String search and manipulation algorithms have been well studied since the beginning of Computer Science as a field. We can see early in the 1960s and 1970s people were using dynamic programming to match two strings [AC75, Bak78, Bir77, FP74, Gal79, Lev66]. In fact, Levenstien [Lev66] published his paper on the edit distance (or Levenstien Distance) between two strings in 1966. In the late 1970s two more well known and more efficient string matching algorithms were developed, Knuth-Moris-Pratt [KJP77] and Boyer-Moore [BM77]. At the same time others in the field were working with suffix trees, Weiner [Wei73] and later McCreight [McC76] present linear time suffix tree constructions algorithms. All of this was done before the concept of making these ideas applicable to biology.

Work is still being done in the area of classical string matching and search. As databases grow larger and the amount of information on the Internet increases more algorithms are needed to index, process and search. Ukkonen published his algorithm for suffix tree construction in 1995 [Ukk95], and it has become the state of the art for not only tree construction but for search. Manber and Myers developed suffix arrays in 1990 [MM90]. Work is also being done on searching compressed elements [AN09, FGN09, CN09, ABF94, DNZ98]

Another type of search is of course regular expression[Brz62, Brz64, MY60, Tho68] which has its roots in S.C. Kleene's automata theory [Kle56]. Regular expressions use a modified version of a state diagram to search for a pattern in a string and are used in applications thought computer science [Aho90, Joh79]. Regular expressions also have uses in searching biological sequences [Bai92, Bai93, KLS92, Sta91, Ste91], but this is not as commonly used as other methods.

### 1.1.2 Beginnings of Bioinformatics

To understand the union of the field of computer science with biology it helps to remind ourselves, of the definition of a string. In computer science theory a string is simply an ordered list of characters over some alphabet. By convention, this alphabet is assumed to be the American Standard Code for Information Interchange (ASCII) alphabet. But all of the algorithms that we have talked about do not take into account the exact alphabet, in fact

most times for examples they used a limited alphabet to show the usefulness of the program. That said, of all of these search, matching and manipulation algorithms are easy applied to nucleic acid strings.

Dan Gussfield talks about his struggle to bridge the gap between these two disciplines in the preface to his book *Algorithms on Strings, Trees, and Sequences* [Gus97]. But we can go back even further to Sankoff and Kruskals book from 1983 [SK83] where they first explore these fields in combination. Though the ideas where present from the early 1970s when Beyer published his paper on the subject [BSS74], the field did not really have any major impact on the community until the early 1980s.

As sequencing techniques became common in biology, it was realized that by finding homologs, or similarities in sequences, you could trace the ancestry of a species. It was further discovered that this could be done computationally, using computers. This was truly the birth of what we now call computational biology. Additionally, this is what was the primary focus of computational biologists was and for the most part still is. Taking the data that comes out of the sequencing machines, and matching it with other data that is similar.

Homology search is the task of finding new locations of sequence that are biologically similar but are not necessarily sequentially similar. This can be seen in the most simplistic form by examining the conversion of DNA into functional protein. The idea is that even though a set of data encodes the same protein it may have a different underlying sequence. To take it one step further, making the assumption that the two, or more, species or sequences in

comparisons have the same common ancestor can we find the differences and calculate how far back they diverged.

Again, as sequencing becomes cheaper this problem becomes even more interesting and broadly defined. We saw this with the boom in algorithms of this type in the 1980s, dealing with small sequence samples; and we will see this again as large sequence samples of different populations within the same species become available in the coming years.

### 1.1.3 Non-coding RNA

Just as the area of string search and manipulation in the computer science realm has evolved from simple string match and distance algorithms to more complex search and pattern matching; so too has the biological side of this discussion.

We all learn the cellular cycle in introductory biology: DNA get transcribed into RNA, that RNA then get decoded into proteins, those proteins then perform functions within the cell. The recent discovery of non coding RNA (ncRNA) and their functions has opened a new field for exploration. By folding in on themselves ncRNA are able to perform functions within the cell without first being transcribed into protein. Knowledge of ncRNA has been present since the early days of molecular biology, but it wasn't until the 1990s that it really became apparent how important a role it played in cell function. Finding ncRNA is still a somewhat

open problem because of the difficulty involved. ncRNA lacks many of the strong markers of other coding RNA, this lack of markers to pinpoint locations creates a problem for search. If the marker is similar to the surrounding DNA, how do you find the marker with high probability?

#### 1.1.3.1 RNA Structure

RNA structure is created when two bases on the same strand of RNA bind with each other. The simplist example is the Watson-Crick pairings which are the A-U and C-G pairs. We call this the secondary structure because it goes beyond the basic structure of the sequence itself. This binding can create intricate patterns (or motifs) that allow ncRNA to have function.

Figure 1 shows some examples of simple secondary structure elements. Shown in the figure are (a) hairpins, (b) bulges and (c) multi-loops. While these are some examples of how secondary structure is formed, they are by no means the only ways RNA structure is created. Not shown are pseudo-knots, which is when structure crosses each other. Most algorithms, including the ones we developed, assume that there are no pseudo-knots in the structure because that makes analysis much more difficult.

Figures 2 and 3 show some examples of sequences with structure. These diagrams show the consensus structure and sequence for tRNA and 5S rRNA respectively. Bases with dots or lines between them are paired. These sequences show some examples of all three elements

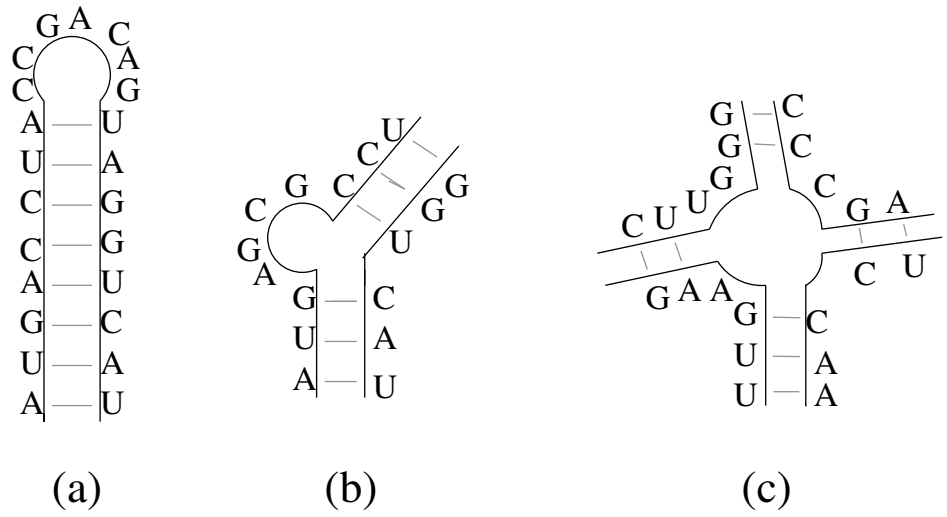



Figure 1: RNA Structure Components Examples.

from Figure 1. Both of these figures were retrieved from the Rfam [GBM03] entry for the respective ncRNA element.

RNA also has the ability to bind on the other edges of the nucleic acid. These are referred to as tertiary structure, because they allow for one base to bind with multiple bases using different edges of the molecular structure. This will not be used in the contents of this paper, so we will not describe it in detail.

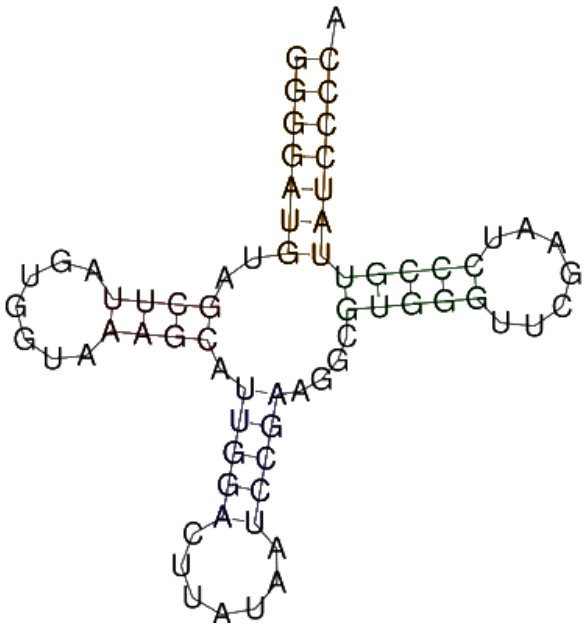


Figure 2: tRNA Structure.

#### 1.1.3.2 Classification

RNA can be classified primarily two ways: structurally and functionally. Structurally there are two major classes, those are internal loops and external loops. Functionally, there are three major classifications; regulation, protein synthesis, and manipulation.

Classifying ncRNA by structure has been well studied [HST06, KL05, LC98]. The idea is one that evolved from the task of protein structure classification [BWF00, MBH95]. SCOR: Structural Classification of RNA [KTH02, THK04] is a database of such structure. The two major classifications of structural elements also have sub-groups. Internal loops are any loops that are between two or more stack, this would include things such as bulges. An

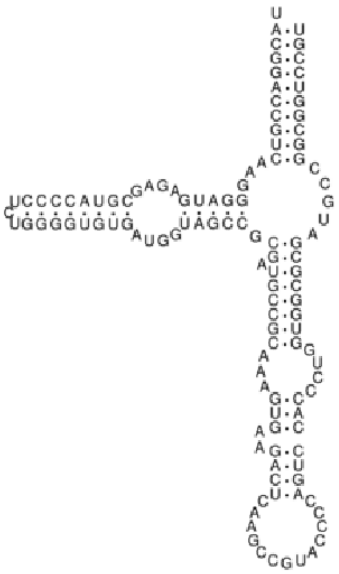


Figure 3: 5S rRNA Structure.

external loop is defined as a loop that is within one stack, a hairpin is an example of this. With the structural elements (or motifs) you can try to determine function with related structures.

Additionally work is now being done on the tertiary structure of RNA and how that affect the function of non-coding RNA.

As far as functions of ncRNA, we see that there are some major groupings; these groupings are the function of regulation. Regulatory RNAs include riboswitches which are found on the mRNA 5' UTR (UnTranslated Region) and microRNA. These functional RNAs work by binding to sites on the DNA to control gene repression and regulation.

RNA is also involved in protein synthesis. Ribosomal RNA (rRNA) and Transfer RNA (tRNA) work in unison to produce proteins in the cell. rRNA decodes the tRNA to create amino acids, these amino acids then get bound together using the data encoded in the mRNA (Messenger RNA).

Additionally, RNA processes other RNA. Some examples are small nuclear RNA (snRNA) and small nucleolar RNA (snoRNA). snRNA are involved in splicing out introns from the strings of mRNA. snoRNA are used in the nucleolus to guide enzymes that modify the RNAs but do not directly do the modifications themselves.

One way to discover new ncRNA involves predicting the structure from newly sequences genomes to see if they are good candidates. Additionally, by aligning known ncRNA segments with other segments of similar function and type the structure can be inferred. The other way to find new ncRNA is to search the entire genome or genome alignment using some known ncRNA profile. We refer to these as *prediction*, *alignment* and *search*.

#### 1.1.3.3 Prediction

The job of predicting new structure for ncRNA is hard and has been accomplished by several programs readily available such as FOLDALIGN [THG07], RNAalifold [HFS02] and others [DWB06, KTK07, KH03, MT02]. Within this category there are three major ways to accomplish the task. The first is to simply predict the energy of the interactions of all

of the bases in the sequence, by trying multiple combinations, you can find the structure that has the lowest residual energy, and this is the most likely. The next two use a multiple alignment to help the previous process, one is to align multiple sequences that do not have structure then predict the structure based on the alignment. The second is to fold all of the input sequences then align them together using the structure and make corrections as you go.

The second major group of methods involves aligning a single sequence to a profile (multiple alignment) of other sequences of the same type. The structure attached to the profile can then be inferred on the new sequence. In this case, it is assumed that a small number of sequences would be experimentally tested to find their actual structure. A profile would be made by hand of these few sequences, this would be the input profile. Programs such as PFastR [ZBA06] do this using dynamic programming, this though takes a lot of memory and CPU consumption to complete.

The last way to computationally find new ncRNA structure is to do a sequence search on an entire genome or genome alignment. This would then lead you to specific areas within the genome that are most likely new locations of ncRNA of the family being searched. This can be done both with only sequence information as well as with given structure information.

### 1.1.4 Thesis Outline

I propose two new algorithms, one solves the second problem above of doing a single sequence alignment to a profile, we also extend the program to create a profile from scratch given only one sequence with structure. We also developed a program to solve the third problem above and search, a genome alignment using the same techniques. In both cases we use the sequence structure to help guide the new alignment, this becomes a constraint on the aliment to make sure the new alignment still folds into a functional unit.

Both works are in either the pre-publication stage or have been published so they are presented separately here, but each assumes some background knowledge on the topic. This background is presented next, followed by the two papers in their publishable form. The two projects are tied together by a common thread, which is RNA sequence alignment using structure information. Additionally, they use the same basic theoretical arguments to perform the alignment. This basic theory is the binarized tree, and the dynamic programming associated with using a binarized tree for alignment. We also present some background on the field in general to help argue why this work is important and what impact it has.

## 1.2 PREVIOUS RELATED WORKS

In this section, we present some of the algorithms that are mentioned during the explanation of the novel algorithms presented later in the text. This should give the reader an idea of the other solutions to the basic problems to be solved. We begin by reiterating the basic edit distance algorithm so that we see what the basic idea of string matching/alignment is. We then profile ClustalW and LARA as they are used as benchmark tests later. Additionally we summarize the algorithms presented for FastR and PFastR as they are the basis for both algorithms here.

### 1.2.1 Progressive Alignment

Progressive alignment construction is a commonly used technique used to create multiple alignments. The idea is to do this by only doing pairwise alignments then constructing the multiple alignments as the combination of the results of all the pairwise alignments.

This is also called a tree-based multiple alignment, or a neighbor-joining multiple alignment. It has been used multiple times in programs such as T-Coffee [NHH00], ClustalW [LBB07], and others [HBS04, SB05, Hol05, DWM06, BKR07]. It is one of the most common methods for producing multiple alignments because of its low computational cost.

The process is recursive in nature, each step builds on the last one. To begin, you start with a single pairwise alignment. This alignment serves as the seed alignment for the entire program. The next step is to do another pairwise alignment between the newly created sequence profile and some other sequence. This is where the variation comes into play; some algorithms align the new sequence to the profile or the multiple alignment, others align to the consensus of this alignment, and still other do more unique things.

However the intricacies of the actual implementation are created, the idea is simple. By building up from a single pairwise alignment, you can create a multiple alignment using only the CPU and algorithm for a pairwise alignment. This is the strategy we implement in PMFastR (Chapter 2)

### 1.2.2 Edit Distance

Edit distance was really the beginnings of what we now know as string alignment. The idea here is to find the number inserts or deletes (*indels*) needed to convert one string, call it $A$, into another string, $B$. This method is also called the Levenstien distance, because V.I. Levenstien was the first to publish on the topic [Lev66].

The method is dynamic programming and has an $O(n^2)$ running time. The dynamic programming table is set up in such a way that each cell $(i, j)$ stores the number of indels

needed to convert $A[1...i]$ into $B[1...j]$ where each is the first $i$ or $j$ characters of $A$ and $B$ respectively. Thus the final number of indels would be the value at $(n, m)$ if $n$ and $m$ are the lengths of the strings input. The actual optimal alignment can be obtained by tracking back though the generated table. Since the calculation of each cell is constant time, we can see the simplify and power of this algorithm.

### 1.2.3 ClustalW

ClustalW is a widely used program for multiple alignment of both nucleotide sequences as well as protein sequences. It is novel in its preprocessing algorithms and it's used of dynamic weight calculation. This algorithm consists of three major steps: pairwise alignment, tree generation and progressive alignment. The pairwise alignments are done to get a distance matrix for all of the sequences being aligned, then the tree can be built from these distances. This tree can be used to progressively build the multiple alignment by first aligning the two closes sequences, then the next closes to them and so on.

The pairwise sequence calculations need to be fast to allow for a large number of long sequences. It also needs to be accurate to get a quality distance matrix for the tree generation step. They accomplish this in one of two ways, depending on the user. The first is to do a simple edit distance calculation, this is slow but accurate. The second is to use a $k$-mer alignment, where each $k$ residues is used as a single character, in an expanded alphabet

and the edit distance is calculated on those compressed strings. In either case the score is simply the identity of the alignment, meaning the percentage of the total alignment that is matching.

The tree construction is a simple neighbor joining algorithm. This means that the lowest distance is chosen as the initial joining point, these two nodes are then combined into a two node sub tree, and the distance between that sub-tree and any other node is the minimum of the distance of that node to either of the nodes in the sub-tree. This is repeated until only one node is left.

The progressive alignment is still the same basic idea presented above, except they make some modifications along the way. Again the alignment at each stage is the same two dimensional dynamic programming presented in section 1.2.2. Each of the closes neighbors is aligned going up the tree. At times it is needed that two previously aligned groups need to be aligned, to score this they simply take the average over all of the pairwise scores for each of the sequences involved at those locations being examined.

The first major item that is novel in their algorithm is sequence weighting; meaning for each sequence involved in the alignment, it has a weight. This weight is for the most part a normalized value of its similarity to all of the other sequences involved. Any sequence that is closely related to one or more sequences has a low weight, one that is divergent has a high weight. This weight is then multiplied into the scoring matrix when the dynamic programming is performed.

The second is position specific gap penalties. The gap opening and extension penalties vary depending on the location in each of the strings. These variations include distance from the beginning and end of the sequences, similarity of the surrounding sequence, difference in the length of the sequence, whether a gap already exists in the alignment being aligned, and others. These improvements help make ClustalW a standard at creating multiple alignments.

### 1.2.4 LARA

Lara uses a combination of a graph building algorithm and integer linear programming to compute an optimal alignment of multiple sequences. The idea is that each base in both (or all) of the strings is a node in the graph. Then edges are drawn to show interactions and aligned bases. This is then converted into an integer linear programming problem and its solution is interpreted as the correct alignment.

#### 1.2.4.1 Graph Theoretical Model

To present the graph construction algorithm we will reduce the problem, as they present it in their paper, to a simple two sequence alignment. It is easy to see how this is extended to a multiple alignment formulation. Each sequence, $A$, is converted into its own small sub-

graph, $(s^A, p^A)$. Here $s^A$ is the set of nodes, simply all of the bases from the sequence $A$; $p^A$ is the set of interactions $(s_i^A, s_j^A)$, where $s_i^A$, and $s_j^A$ represent the $i$-th and $j$-th base in sequence $A$ respectively. This interaction exists if those two bases are said to be pairing in the structural annotation.

The alignment of $A$ and $B$ is a graph $G_s = (V, L)$ where $V = s^A \cup s^B$ and $L$ is the set of *lines* that are matching bases in the alignment. A line is a representation of the matching of two base pairs, call them $s_i^A$, and $s_j^B$. Any subset of $L$, where there does not exists two lines that cross or touch is called a *valid* alignment. Additionally interactions between nodes within the same sequence (annotated structure) have special lines not in $L$ but are also present on the graph, call this set $I$.

To complete the graph gap edges are added with connections such that they would span any area not paired with a base in the second sequence. In reality there is a gap edge for every possible gap that could exist in the alignment. These gaps are again said to be in conflict if they cross or touch. They are connected to all of the bases that would be covered by that gap. This extends the *valid* alignment to say that only one line can be connected to each node other than gap nodes. This way each node in the sequences is either aligned to a base in the other sequence or is attached to a gap. In extension if a gap is connected to by one node, it must be connected to all other which there are lines present.

An interaction match is defined in the text simply as any interaction that is present in both sequences and the ends of the interaction are matched with each other.

Each line and interaction is assigned a weight. This weight comes from the BLOSUM matrix for connections between bases and RIBOSUM matrix for the interaction lines. Additionally a gap has a penalty. The sum of all the lines and gap edges included in a valid alignment is the alignment score.

#### 1.2.4.2 Integer Linear Programming

For the integer linear programming formulation it is straightforward to see that the maximization unit is the score for the alignment. Each line and each gap edge becomes a binary variable. The interactions are split into interactions iterations; that is to say if there is an iteration $I_{ab}$, this would be split into two directed interactions $I'_{a\to b}$ and $I'_{b\to a}$. Then each member in this set of directed interactions is a binary variable in the model.

This model is then put into an ILP solver and the output would be a set of interactions, lines and gaps that is the optimal solution. This can then be re-interpreted as above into a valid alignment. The problem formulation can be seen in [BKR07].

### 1.2.5 CMSearch

CMSearch is a part of the Infernal package. It uses Covariance Models to align an input profile to a sequence. The pre-processing in creating the covariance model allows the search to be very fast.

A covariance model (CM) is similar to a Stochastic Context Free Grammar (SCFG) in that it uses states and transitions from state to state with output at each transition and probabilities on each transition to determine if an input matches the model. How CMs differ from SCFGs though is that in an SCFG the transitions are static for any given state and are not position specific.

To build the CM the first action is to create a guided tree for the profile. To do this they produce a consensus structure and sequence from the input; this involves removing columns that have less than a certain percentage gaps, or using an HMM probability score. Once a consensus structure and sequence are determined, they begin to build the tree. For the most part the tree is built upon the structure. The nodes of the tree are similar to nonterminals in the CFG only with one type of transition. Each has an output on the left or the right (or both or neither). It also has a specific type. For instance you always start with a ROOT node, any pair of structure is always a MATP node. The construction here is similar to that presented in section 1.2.7.1.

The final step in creation of the CM is the actual conversion of the guidance tree into the CM itself. Each node type gets a set of states that it can transition through. This is always some subset of Match Pair, Match Left, Match Right, Delete, Bifurcation, Insert Left, Insert Right, Start, and End. The insert left and right states are special because they are self pointing, meaning if you transition to that state you don't leave the node. Additionally the Bifurcation state is only used by split nodes, and it in turn calls two start nodes for its two children.

Once the CM is created to use it for search a divide and conquer strategy is implored. They used what they call the CYK/Inside and CYK/Outside strategy. Here CYK stands for the Cocke-Younder-Kasam algorithm. This is similar to the Myers-Miller [**?**] algorithm for divide and conquer. The main concept is that if you are performing an alignment using a DP table, then you know at some point the optimal solution must pass though each row, and each column. Therefore, if you calculate the sub problems for the left of that column and the right of that column you can find the row in that column where the optimal passes though. The main idea is that you can divide the problem into smaller, tractable sub problems without the loss of any information along the way.

The inside/outside algorithms they describe needs the state $v$ from the CM to be given to calculate the optimal locations in the search string where the optimal alignment would pass. To guarantee that the optimal alignment would actually pass though the state we would calculate this on, they use the bifurcation nodes. By starting with the top bifurcation node

you hope to divide the problem as close to in half as possible, you then do this on down the CM. Once you have split at all of the bifurcation nodes, what are left should be small enough problems that you can brute force the answer to the small sub-problems with low computational cost.

### 1.2.6 BLAST

BLAST, or Basic Local Alignment Search Tool, is a simple yet effective and efficient search tool for both protein sequences and nucleic acid sequences. The idea is simple, find short exact matches from the query in the target, extend the exact hits till the score cannot be improved, if the score is above a threshold output.

The algorithm uses a scoring metric called maximal segment pair (MSP). This is simply the highest scoring identical length segments across two strings. The score is a simple scoring matrix, in the original paper [AGM90] they used +5 for matching bases and -4 for mismatch. This has since been extended to use more traditional, calculated scoring functions. Because this metric is so elementary, and because of the restriction that the sub-sequences be the same length, the calculation of the score is linear with respect to the segment length.

The assumption is that when a search is preformed the user will input some search boundary for the cutoff of the score, $T$, which they would like presented. Using this, and some pre-

determined, ad-hoc, word length $w$ then calculate all of the feasible words of length $w$ that would lead to an alignment of score $T$ or more. These words are then the index to a linked list of all the location in the query where the corresponding word resides. This leads to a constant time lookup (assuming $w$ is constant) for each $w$-mer in the search string.

The next phase is extension. This step is simple because of the first step. Here they start in one direction and add on one base at a time, adding or subtracting from the alignment score, the base score of the two residues in the target and query that they just added to the alignment. This is a constant time operation. They stop this when the total score drops below some threshold beyond the best score they have seen so far in that direction. This is as simple as saving the max when you see it.

After the extension has reached its maximum in both directions, you have found the local maximum for that "seed" word in the target. You can then check the score you achieved to see if it is above the global threshold score.

It is easy to see how this search method would be very fast after the initial preprocessing, that is why it is very widely used. They have also released a follow up to this algorithm which allows for gaps and profile in the alignment this is called PSI-BLAST and Gapped-BLAST [AMS97] but still follows this same basic principle.

### 1.2.7 FastR

FastR or Fast RNA alignment was published in [ZHE05]. This was the major influence and basis for the two works presented here. we detail the algorithm here, so that in the texts that follow a basic understanding is had of the improvements made in each of the two algorithms presented. In summary FastR created a binarized tree of an input RNA sequence with structure and uses this tree to guide the alignment with another sequence without structure. The output is a global alignment of the two input strings. This is done with structural conservation consideration and as an added benefit is the induced structure from the input profile is projected onto the target sequence.

#### 1.2.7.1 Binarized Trees

The idea for the Binarized tree was first discussed by Bafna *et al.* in [BMR95]. They describe the fact that using a tree representation of the structure you can infer the structure onto the newly aligned sequence. They include only those elements of the query in the tree that correspond to structure. They also only introduce the notion of adding extra nodes to make the tree binary in nature. An expanded formulation appeared in Zhang *et al.* (2005) [ZHE05] and this is the expanded representation.

We assume that pseudo-knots do not exist in the RNA sequences that we will examine. That is to say, we assume that for any pair of bases, there is one parent set of bases that exist such that no other pair of bases cross. Mathematically, if we are examining paired bases $(i, j)$, it has a parent $(i', j')$ such that there is no other pair $(i'', j'')$ where $i < i'' < i'$ or $j' < j'' < j$. With this assumption made we can represent the structure of any ncRNA as a tree were the root is the outermost parent of all paired bases and each node represents a base pair. From here on we will call this set $M$ .

The problem is that each node may have a multitude of child nodes, thus the tree is not easily traversed. Additionally, not all bases in the RNA sequence are represented in the tree. To be able to use the tree for the basis of an alignment these two factors need to be corrected. Zhang *et al.* present an algorithm in [ZHE05] to binarize the tree and include all the bases from the sequence. This tree is called $M'$.

In this new binarized tree there are three types of nodes. First are the nodes that existed in the original structure tree $M$ , these represent the paired bases, including their location in the RNA query. These are referred to as *solid nodes*. The second is a node that represents an unpaired base, these are called ,*dashed nodes* but they have only one child. The last is a dashed node that has two child nodes. These represent points where two structural elements occur in parallel.

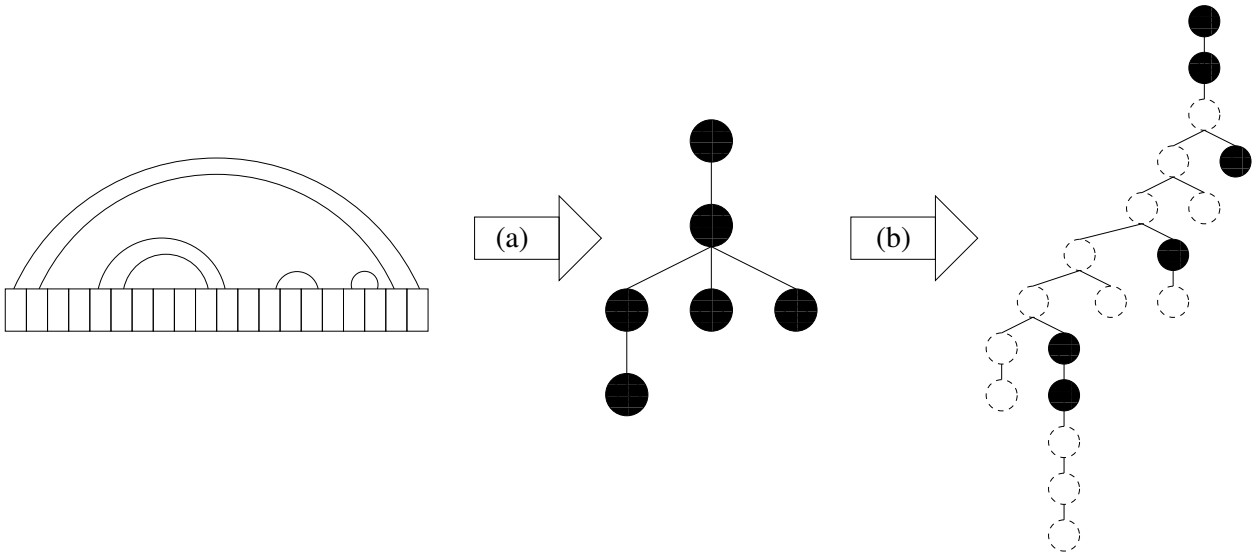



Figure 4: Binarized Tree Example.

Figure 4 shows the conversion of an ncRNA into a binarized tree. We can see that (a) shows the conversion from sequence to the structure tree $M$; (b) shows the binarization results, it is obvious here how the three types of nodes work to represent the entire sequence.

#### 1.2.7.2 Alignment Procedure

Again the alignment procedure was originally published in Zhang, *et al.* [ZHE05]. The central theory is that the search is done by traversing the binarized tree $M'$. For each node, call it $v$, which represents some subset of the query sequence say the end points of that substring has end points $(l_v, r_v)$. The search is then saved into a dynamic programming table. The DP table is three-dimensional, in particular it is of size $|M'|$ x $n$ x $n$, where $n$ is

the length of the target sequence. we call this array $A$ and it is indexed by the node $v$ and the matching locations $(i, j)$. Here $i$ and $j$ represent the locations in the target that align with the locations at the two end points of the node $v$.

A modified version if the procedure is shown in Figure 6. In this procedure $\delta$ represents the score associated with the query and the target at these locations matching the same structure. Similarly, $\gamma$ is the affine gap penalty assigned to any unmatched bases in the alignment. We see that each of the three types of nodes is handled differently. If the node encodes structure, then we not only test for inserts, deletes and matches in the sequence but also give some scoring consideration to structural biding conservation. It is relevant to point out that $\delta$ takes into consideration the covariance mutations that may occur between sequences. For nodes with two children, meaning this is a branching point, because no subsequence is directly encoded by this node the score is simply the best combination of scores from its two children without a gap between them. This fact is key to the multi-threaded improvements made later.

#### 1.2.7.3 PFastR

In Zhang *et al.* (2006) [ZBA06] they discuss the profile based version of the previously stated algorithm. The program developed is called PFastR, or Profile based Fast RNA alignment. The main improvement to the algorithm was that it could take in a profile with structure to

a single sequence without structure. This involved using a Position Specific Scoring Matrix (PSSM).

To accomplish this task they redefine the scoring functions $\delta$ and $\gamma$ based on the location within the profile that the item was being aligned to. Because each location is now not a single base, but a collection of bases that could vary and be a point of a SNP, they calculate what is essentially a probability of it being that particular base that that position. This allowed location that where of high variance to not greatly degrade the alignment score and be detrimental to the alignment.

### 1.2.8 Rfam

Rfam or RNA Families is a database of ncRNA. Each RNA family, of which there are now 1372 (as of version 9.1, January 2009) has an alignment with associated structure.

The first version of the database was released in 2003 [GBM03]. This is a repository for almost all known, if not all known ncRNA sequences. The alignments and information is all community submitted, so the sources for each alignment are different.

Most families include some background including a structure diagram. More importantly, each family contains two multiple alignments: seed and full. The seed alignments are hand curated and much smaller. These seed alignments are then used along with CMSearch and

CMAlign of the Infernal package to search nucleotide databases. The results of this search are presented as the full alignment. The search database is based on EMBL nucleotide database [SBB02]

# CHAPTER 2:
# PMFastR

This chapter contains a previously published work [DBZ09]. we co-authored this paper entitled *PMFastR: A New Approach to Multiple RNA Structure Alignment* with Jocelyn Braund and Shaojie Zhang. It is to be published in the proceedings of The 9th Workshop on Algorithms in Bioinformatics (WABI). It is presented here for the most part in-print with a few exceptions. Several sections and figures were left out and put into the supplemental material due to space limitations, but have been included here for reference.

## 2.1 INTRODUCTION

A high quality multiple alignment of RNA sequences is crucial for RNA homology search [ZBA06], structure analysis [ED94] and discovery [RE01, WHL05]. Even though methods for multiple alignments of DNA and protein sequences are well studied [DMB05, Edg04, NHH00, THG94], multiple RNA structure alignment is still an open problem.

Aligning two RNA sequence with consideration of the structural information comes as an extension of the RNA structure prediction studies [JTZ89, KH03, ZS84]. 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 [BKR07, JLM02]. 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 [San85]; this is the RNA *sequence-sequence* alignment problem. Another potential input is to have a structural profile or other structural information for one of the sequences, but not for the second sequence; this is the *sequence-structure* alignment problem. Several methods have been developed for this problem and are used for finding RNA homologs. FastR [ZHE05] and PFastR [ZBA06] use a guided trees and dynamic programming to a globally align a sequence or profile to a given target sequence. CMSEARCH [WR04] and RSEARCH [KE03] 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 [JLM02]; 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 [HBS04], MARNA [SB05], Stemloc [Hol05], STRAL [DWM06], and LARA [BKR07]) use pair-wise RNA alignment (sequence-sequence or structure-structure) for all sequences and then combine these alignments into a multiple alignment using T-Coffee [NHH00] 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 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. 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 [ED94] accomplish this 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 [GMM05] 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 curating 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 5. 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

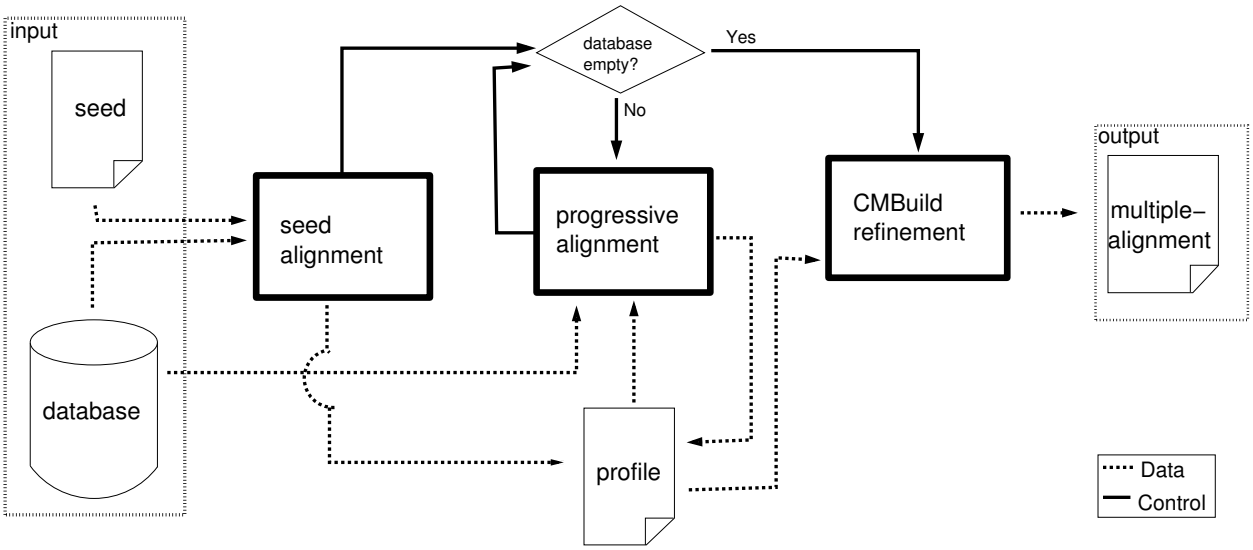



Figure 5: Overview of PMFastR.

aligns the rest of the input data set. Finally, we run the CMBuild refinement program to improve the unpaired gap regions in the alignment. 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. It requires as input only one sequence with structure and multiple sequences without structure 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 section, we run several benchmarking tests on PMFastR with other multiple RNA alignment tools using BRAliBase (version 2.1) [WMS06] as test data sets, which are extracted from Rfam database. We also regenerate all Rfam hand-curated seed alignments (version 8.1) [GMM05] automatically with comparable results.

## 2.2 METHODS

PMFastR is based on the methods used in FastR [ZHE05] and PFastR [ZBA06] 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.

### 2.2.1 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 [ZHE05]. The number of nodes in this tree is $O(m)$, where $m$ is the number of columns in the profile. Figure 6(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. (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 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)

Figure 6: PMFastR Alignment Algorithm

### 2.2.2 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 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. Figure 7 shows that FastR allocates an array $A$ which is of size $n * n * m$ whereas PMFastR only allocates an array of size $band * band * m$ where $band$ is some banding constant and $m$ and $n$ are bounded by the lengths of the profile and target respectively. Figures 6(b) and 6(c) 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. They are added back when the profile is output, by adjusting the quality criteria we can also affect the length of the profile used for alignment.

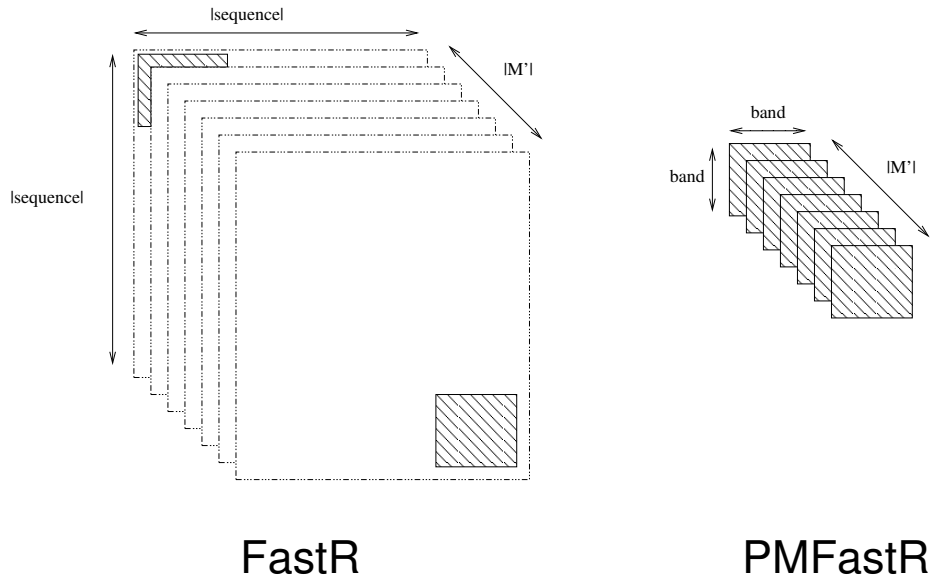



Figure 7: Theoretical Memory Comparison

### 2.2.3 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 6(a) 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.

```
 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
```

Figure 8: The Multithreaded Design

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 8 shows the node processing procedure of the improved algorithm. Because the threads are queued, we can control the amount of simultaneous computation. 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.

### 2.2.4 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 in ClustalW [LBB07] and use the ClustalW guided tree to direct the order of the input for PMFastR.

Figure 9 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 [Edd] 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.

Figure 9: The Multiple Alignment Procedure

## 2.3 EXPEREMENTAL RESULTS

We evaluated the performance of PMFastR based on memory consumption, running time and quality of the alignments. First, we compared the memory consumption of PMFastR with that of its predecessor FastR alignment procedure, these results are shown in the supplementary data. Second, we looked at the improvement in the running time of the multi-threaded version of PMFastR. Finally, we compare the performance of PMFastR to five other multiple alignment tools [BKR07]. We also generate multiple alignments for every RNA families

from the Rfam 8.1 database [GMM05] and compare them with the manually-curated seed alignments.

### 2.3.1 Memory Consumption

To test the memory consumption, we ran both PMFastR and the original FastR alignment procedure with identical input pair sequences. In Figure 10 the memory consumption is represented on the y-axis in function of the sum of the length of the two input sequences on the x-axis. The top line (.) is the original FastR algorithm and the bottom line (+) is PMFastR., we used an input dataset of 3000 random pairings of Cobalamine sequences. The individual sequences range in size from 167 to 298 bases. All of the structures are available and the sequences were aligned to each other with equal probability. The regression was drawn using MatLab on a cubic regression model for the memory consumption of each algorithm. It can be observed that the original algorithm is cubic in space consumption, and PMFastR is linear as hypothesized. Though the goal dataset is 16S rRNA, we could not test this benchmark in a real environment with the resources available as FastR has problems handling such large sequences. We assume that the regression would still hold, since similar results were achieved for both 5S rRNA and tRNA. We also build a multiple alignment of 567 16S rRNA sequences using PMFastR, which is available on the supplementary website.

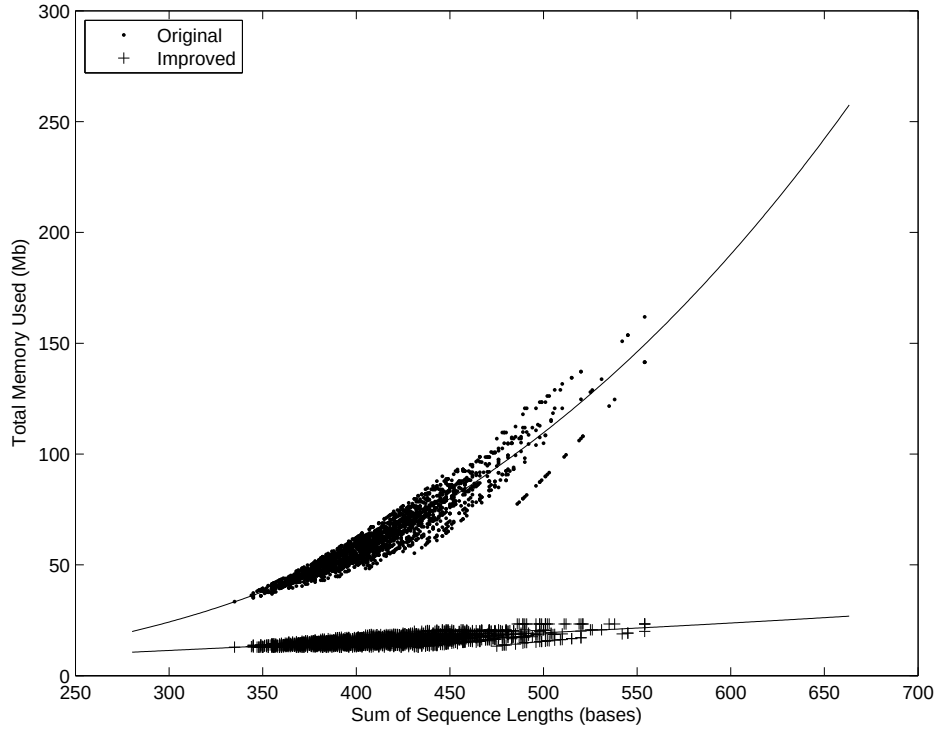



Figure 10: Memory Consumption

### 2.3.2 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 and 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

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 |

increase is minimal. This is due to the dominance of communication overhead as the load on each thread diminishes.

### 2.3.3 Overlapping inferred structure

We compared the alignments generated by PMFastR to those created by ClustalW using overlapping inferred structure. This metric was created by generating a multiple alignment using ClustalW and PMFastR. The output of ClustalW has no structure, so the first sequence was chosen in the output as the *pseudo-seed*. The *actual* structures from CRW are available for all sequences in the set, so the original structure of this sequence was attached to the alignment. The next step was to take the two generated profiles (alignment plus structure) and separate out each sequence individually. Each sequence was then compared to the original structure provided by CRW to see how many of the stacks contained in the inferred structure overlapped.

Table 2: Overlapping Inferred Structure Results

| Set Type | PMFastR | ClustalW | Number Of Sequences | Average Pairwise Sequence Identity |
|---|---|---|---|---|
| tRNA | 97.02% | 90.20% | 1088 | 23.79 |
| 5S rRNA | 64.68% | 35.25% | 1852 | 62.82 |
| 16S rRNA | 75.73% | 61.75% | 567 | 48.36 |

Figure 12 shows some examples of what is and what is not considered an overlap. The two examples on the left side (a) and (c) are overlapping structures, while the examples on the right (b) and (d) are not overlapping. To be an overlapping structure both stacks need to have at least one matching pair in the same position. Table 2 shows the results of this test on varying set types for both ClustalW and PMFastR.

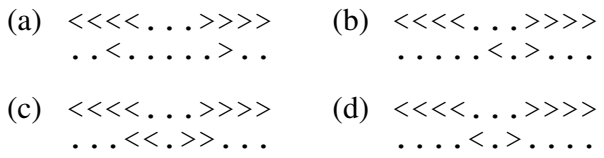



Figure 11: Overlapping Inferred Structure

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

We first choose the widely used benchmark set BRAliBase 2.1 [WMS06] as a out test set, which is based on seed alignments from the Rfam 7.0 database. We compare PMFastR to four other structure-based RNA alignment tools (LARA, FOLDALIGNM [THG07], MARNA, and STRAL) and one purely sequence-based multiple alignment tool (MAFFT [KKT05]).

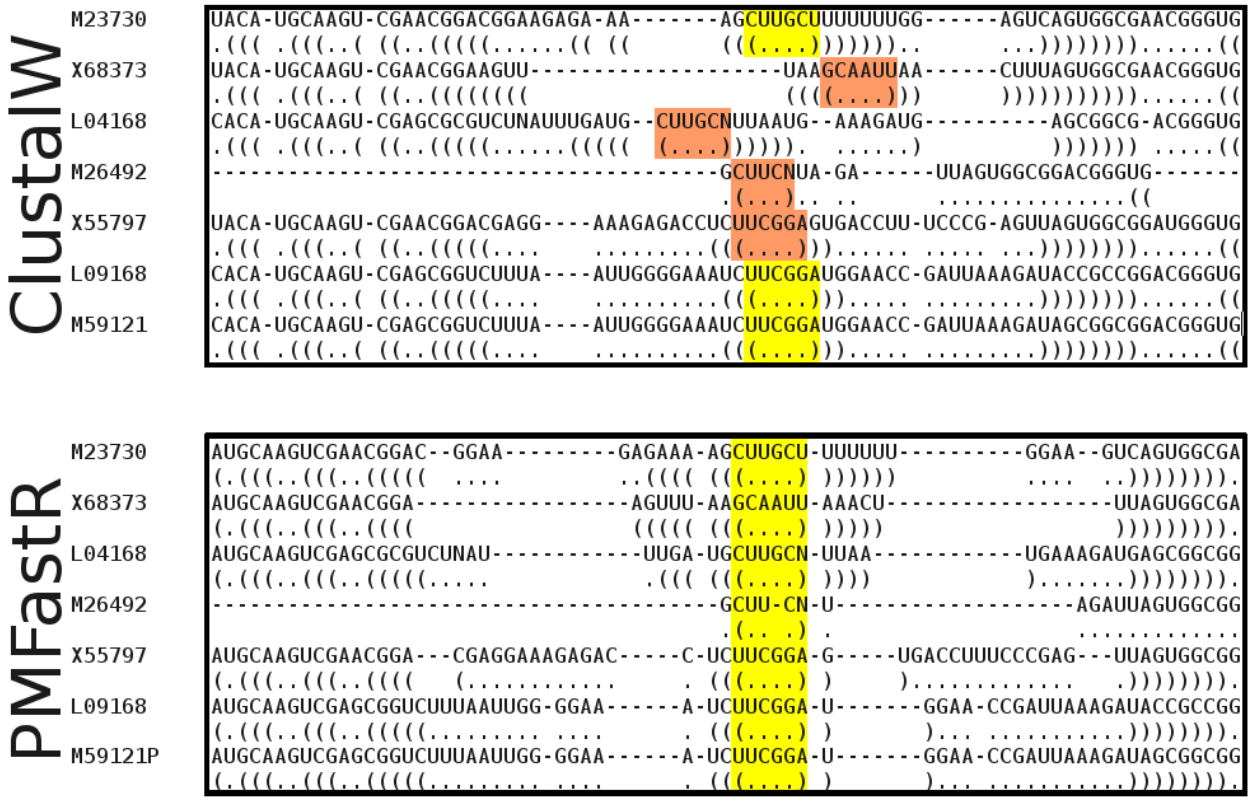



Figure 12: Examples of Corresponding Alignments

In order to compare the alignments generated by PMFastR with the alignments generated by these other tools, we use 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 [GWW05, TPP99]. It is similar to Compalign which has been used in LARA's benchmarking [BKR07]. 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) [GWW05, WHL05] with

the differences that SCR rewards the compensatory mutations and uses reference structure from the benchmarking alignment while SCI first uses RNAalifold [HFS02] 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.

To remain consistent with Bauer *et al.* [BKR07], Figure 13 and Figure 15 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.* [BKR07]. 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 17 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 [CSS02] with average pairwise sequence identity of 48%. The alignment took

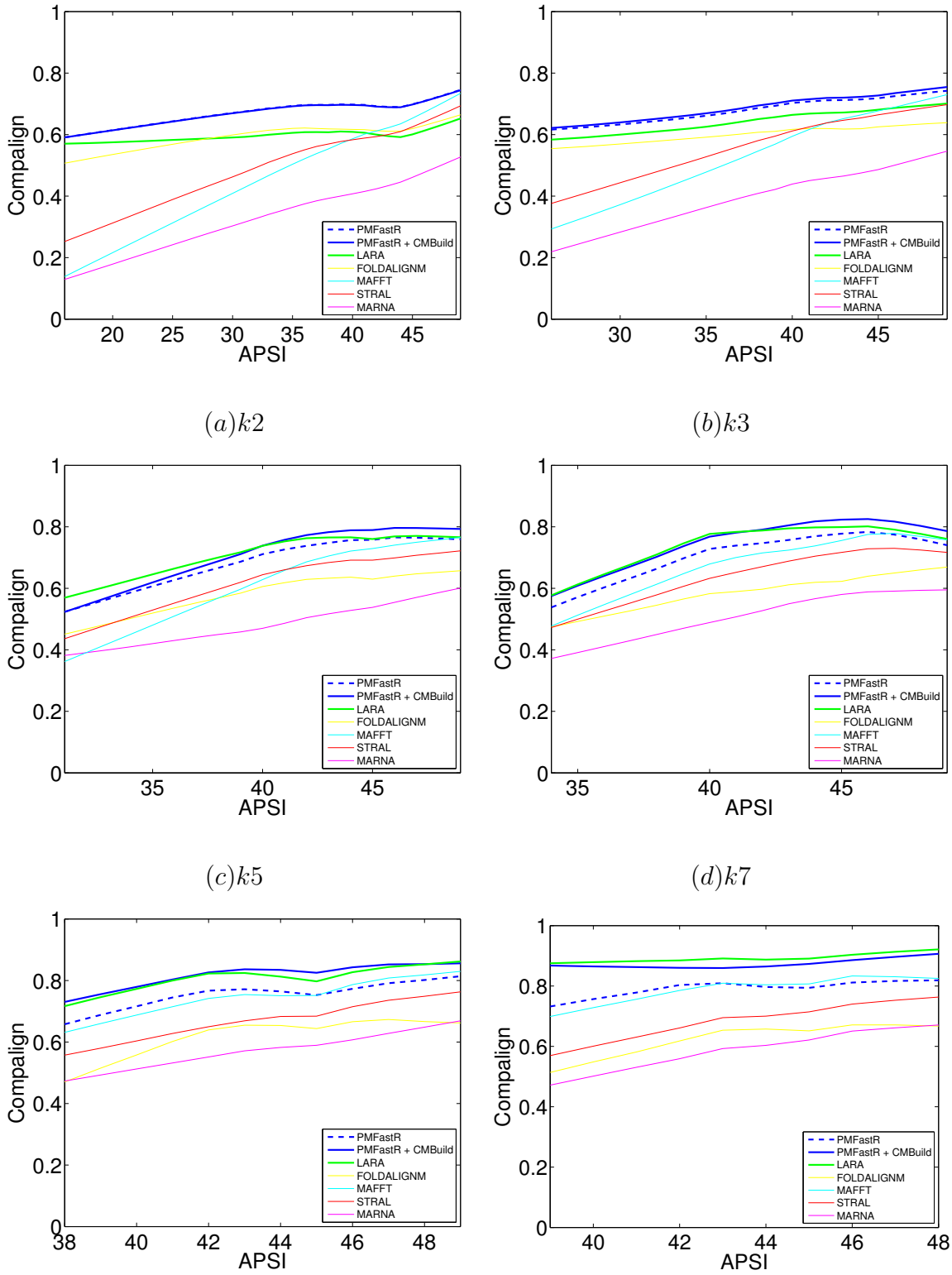
$\sim$40 hours, and had an average maximum memory consumption per alignment of 3.63 Gb. This is the first multiple alignment, to our knowledge, of 16S RNA sequences which takes into account secondary structure information. This alignment is available on our supplementary data website.

## 2.4 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 were 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 allowed 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 allowed us to run it in a multi-threaded manner, thus drastically reducing the wall time. These modifications did not alter the quality of the algorithm's results, and we showed that PMFastR with CMBuild refinement does better than other state-of-the-art RNA multiple alignment tools and created 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.

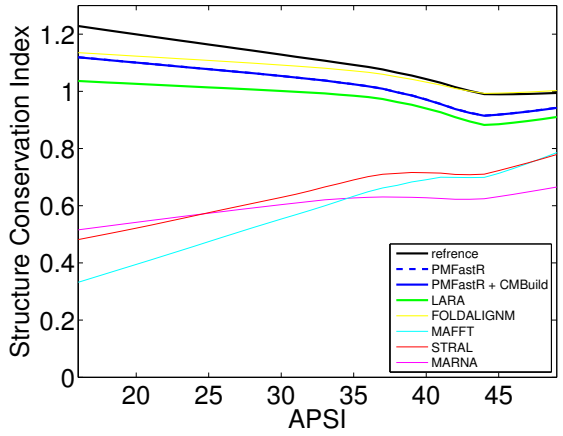



$(a)k2$ $(b)k3$

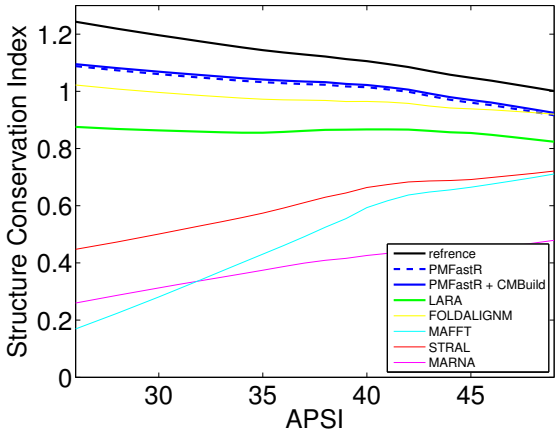
$(c)k5$ $(d)k7$

$(e)k10$ $(f)k15$

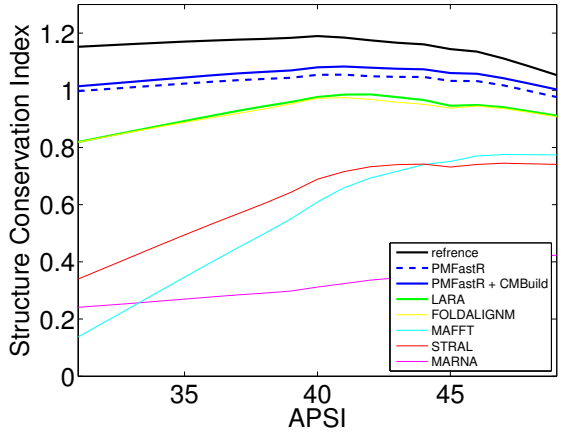
Figure 13: Compalign Benchmarking.



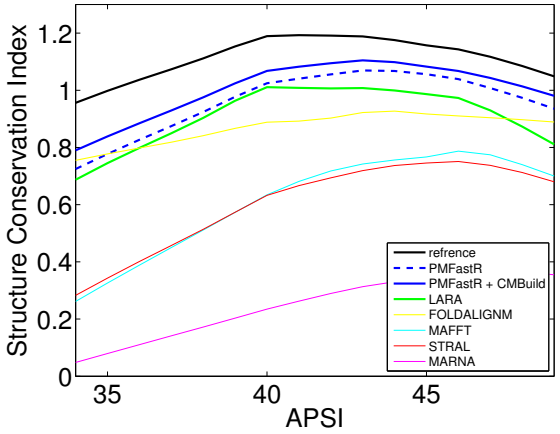

$(a)k2$



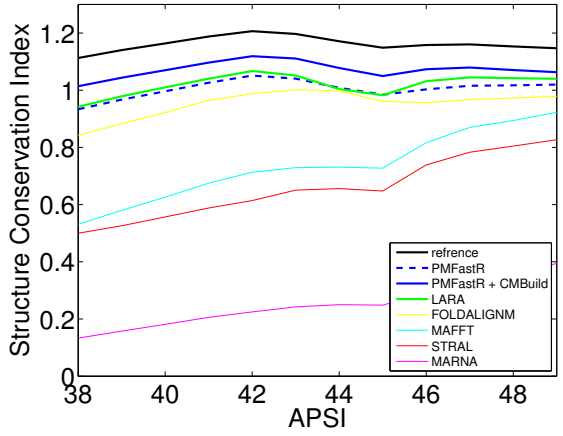

$(b)k3$



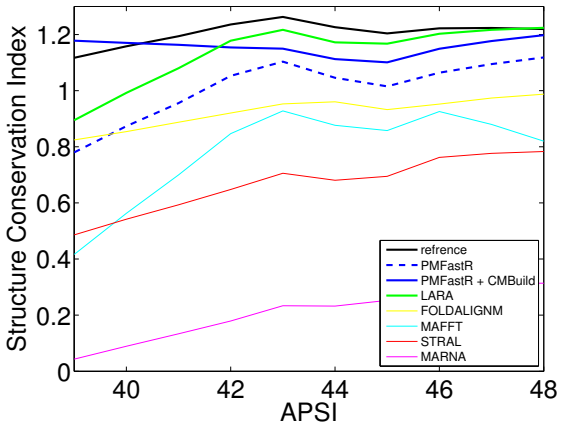

$(c)k5$




$(d)k7$

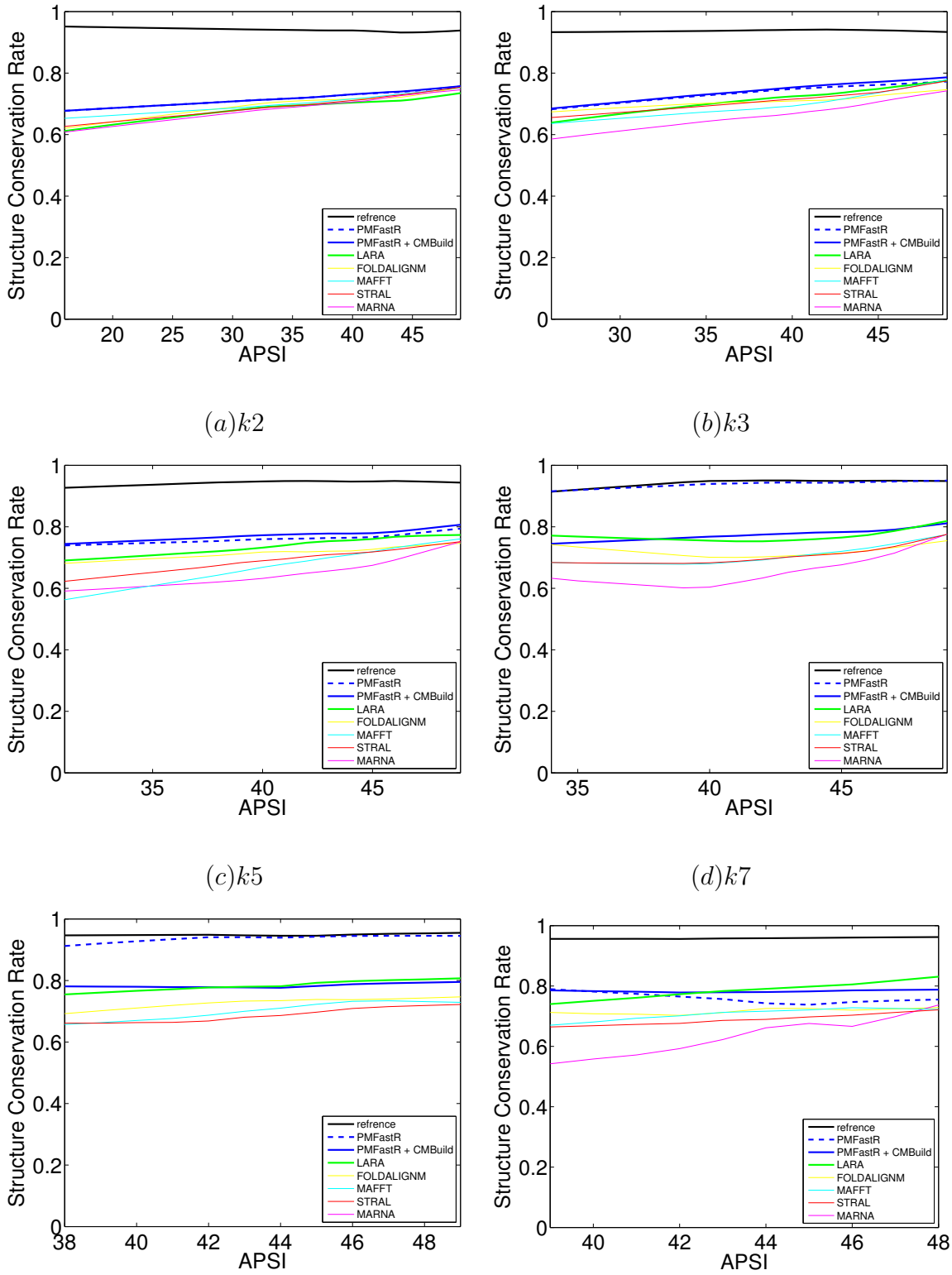



$(e)k10$




$(f)k15$

Figure 14: SCI Benchmarking.

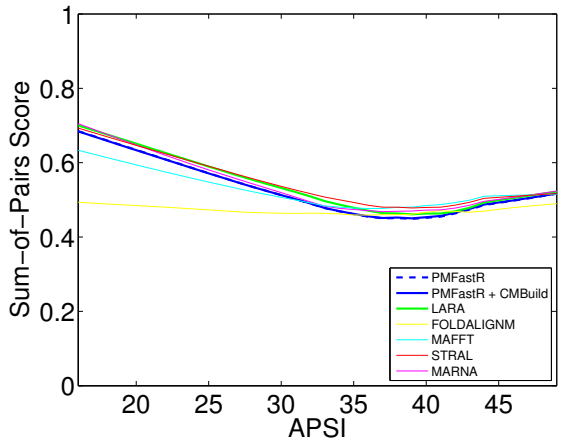



$(a)k2$ $(b)k3$

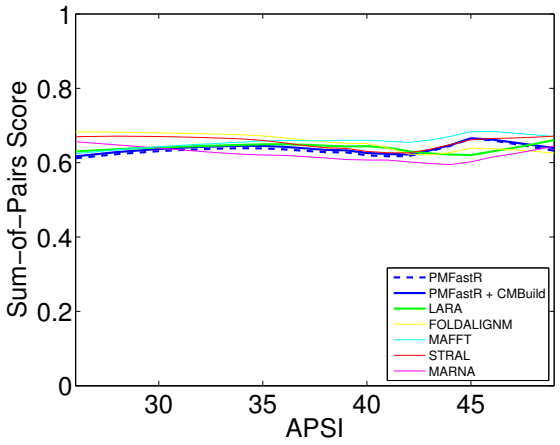
$(c)k5$ $(d)k7$

$(e)k10$ $(f)k15$

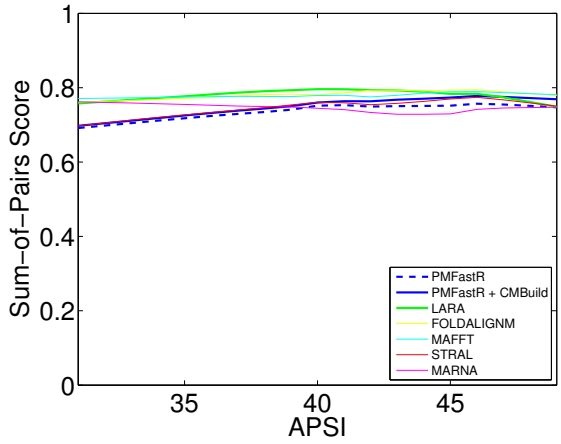
Figure 15: SCR Benchmarking.



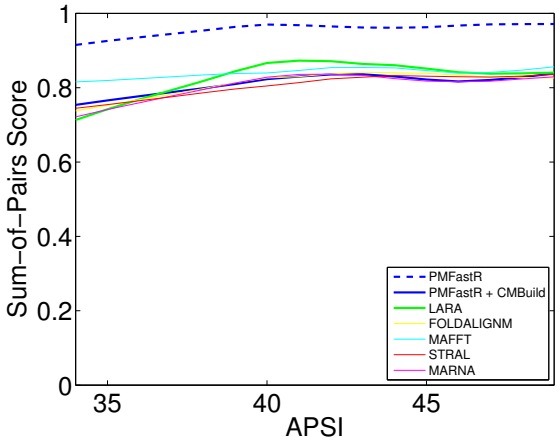

$(a)k2$



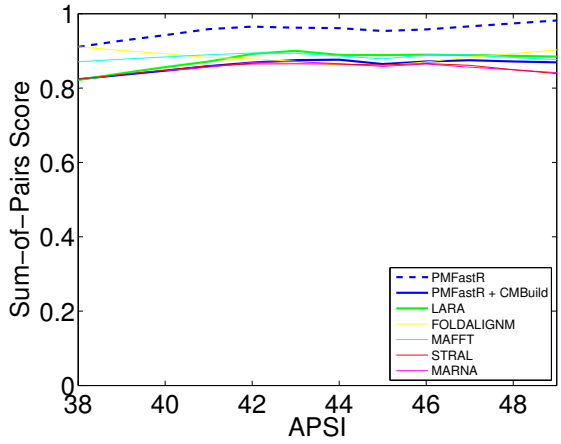

$(b)k3$



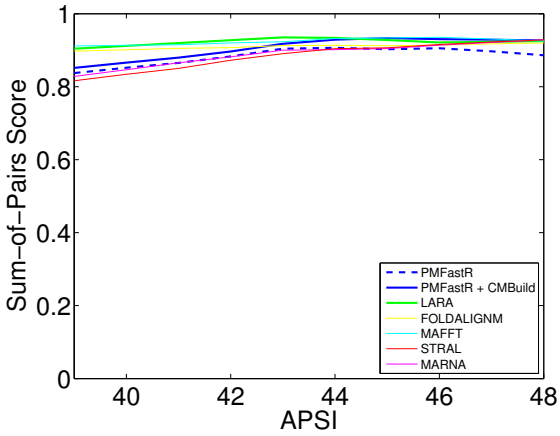

$(c)k5$




$(d)k7$

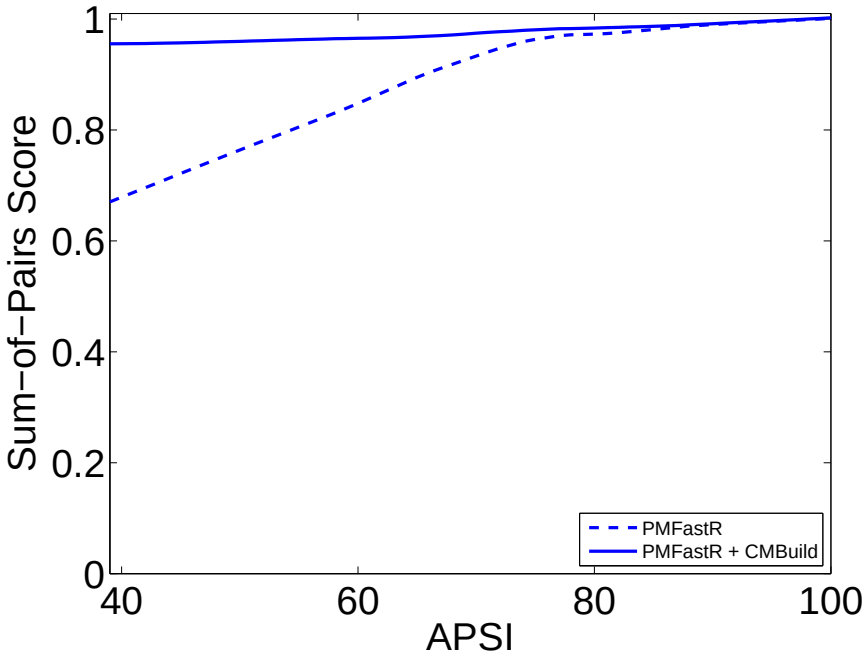



$(e)k10$

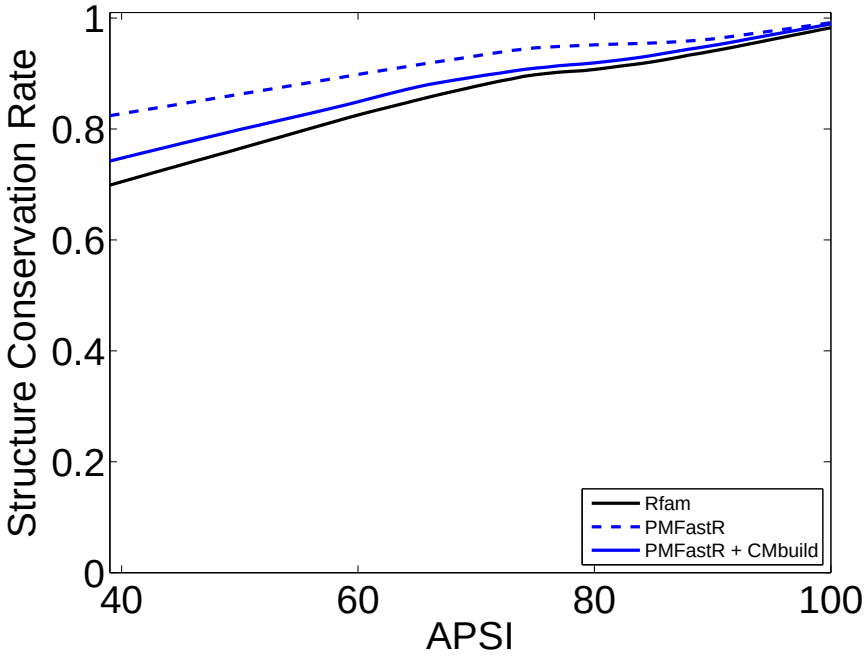



$(f)k15$

Figure 16: SPS Benchmarking.




(a)




(b)

Figure 17: Rfam Comparison

# CHAPTER 3:
# AlnAlign

Here we present an method we call AlnAlign. This project is still in the conceptual state, we present it as future work. The concept is simple, since we know we can align a profile to a sequence using structure, can we align a profile to a genome alignment. This should provide more information about the RNA structure in that critical elements such as ncRNA should not be lost in evolution. To this affect, the structure should be conserved between species. Using this we now apply the alignment procedure above to a multiple sequence alignment and use it to search a genome. With the addition of a bonus for covariance mutation to discover conserved structure.

## 3.1 INTRODUCTION

The problem of aligning alignments stems out of the need to build better multiple alignments, though the uses and techniques have expanded considerably recently. In many cases a multiple alignment is constructed progressively[NHH00, LBB07, HBS04, SB05, Hol05, DWM06,

BKR07], meaning in the most basic case, each sequence is aligned to only one other sequence or to a pre-existing multiple alignment. This new sequence then gets appended to the end of the other sequences and the process is allowed to run again until all sequences have been aligned.

More complex multiple alignments construct a tree of all of the single sequences, then align the sequences according to the tree. First aligning the two closes sequences (however *close* is defined), then the next closest to that and so on. In some algorithms they do not pick the closes one sequence to these newly aligned sequences, they simply pick the closest two sequences remaining. This means these algorithms get to a point where they would need to align two of these sub-alignments together [LBB07].

The problem within aligning alignments that truly differentiates them is the calculation of the gap cost [KS04]. It has been shown in two sequence alignment that a linear gap cost calculation provides the best results than a constant gap penalty. When you expand this linear gap cost calculation to a two alignment problem is becomes NP-complete [Kec93, Mai78, WJ94, War95]. To solve this most algorithms simply do not use linear gap calculations. This reduces the quality of the alignment but can greatly lower the running time as well.

The other solution is to develop some sort of heuristic for this. Kececioglu and Starret [KS04] present a heuristic that is tractable in practice, it is still NP-complete in the worst case. Another algorithm called MSA [GKS95, LAK89] also uses heuristics to solve this issue but they do not compute the *exact* gap penalty, but they make an overestimate.

The basic assumption when working with aligning alignments is the quality score. The score used by most programs is the sum-of-pairs objective [CL88]. The most simple calculation of this is for each column, for each pair of sequences, how many match. This is where they determination between linear-gap scoring and simple scoring is shown. For a linear gap scoring methodology, its not simply the number of matching pairs, but for each non-matched pair, is it within some pre-existing gap or is it opening a gap. The calculation is similar to the affine gap penalty calculation seen in two sequence alignment, where each gap has an opening score, then an extension score. The total penalty of the gap is the length of the gap times the extension score plus the gap opening penalty.

While the initial purpose of the study of aligning alignments was to improve the creation of multiple alignments, there are also other uses for these algorithms. We already know that by searching a single genome we can find new functional elements and new information that may be hidden inside a specific species' DNA. The newly available multiple genome alignments should contain more relevant data than just a single sequence. By examining the entire alignment together it should be easy to see how covarying mutations are present within multiple species and this will provide more detailed information about the newly discovered or even already known features.

The problem is that even the fastest algorithms for aligning alignments are still not efficient enough to run on whole genomes in a tractable amount of time. Those that are focus on proven structure. We present here a new algorithm, designed to search a multiple genome

alignment using RNA sequence structure information. Not only that but it is able to performs this search in a tractable amount of time. The algorithm behind the alignment is similar to that seen in PMFastR. We then make some adjustments and improvements to make it able to align to multiple alignments, these are the adjustment of the algorithm to search for local alignments, the scanning procedures and of course the multiple alignment position specific scoring.

Many of the currently available alignment-alignment programs deal primarily with proteins. The interactions of proteins are more complex and intricate structure and interactions. While the basic ideas are similar for ncRNA, there are differences that make it more tractable. Additionally since the knowledge of improving RNA alignments is present, it should be able to transition this.

In this paper we present the algorithm itself as well as some comparisons to other genome search algorithms. Section 3.2 describes all the methods used to convert PMFastR into an alignment-alignment algorithm. Then in section 3.3 we conclude and discuss the future of this project.

## 3.2 METHODS

The alignment method used is similar to that presented in PMFastR [DBZ09]: a guided tree along with dynamic programming are used to find the optimal alignment. For AlnAlign the procedure was altered to generate an optimal local alignment rather than a global alignment. All of the methods described in [DBZ09] still apply. This section begins with a review of the procedures from PMFastR, then continues with a description of the transition to using a multiple alignment as the query, then finally the scanning procedure is introduced to show how AlnAlign can be used to search a genome.

### 3.2.1 The Alignment Procedure

A structured RNA profile can be represented as a set of paired bases. Let us call such a set of paired bases M . Let us also assume that there are no pseudo-knots, or crossing bases in that set. It can then be said that each pair of bases $(i, j) \in M$ either is the outer most base pair, or has a unique parent $(i', j')$ such that there does not exist another base pair in $M$ where $i < i'' < i'$ or $j' < j'' < j$. Using this nesting, or parental, relationship we can generate a tree out of the bases in $M$ . This tree can then be adapted to include the unpaired bases as spurious nodes and dividing nodes to represent parallel pairs, such that the tree will be binarized. This new tree will be referred to as $M'$. The procedure to create the tree is the

same as the one used in FastR [ZBA06]. This binarized tree can then be used to guide a dynamic programming alignment.

All of the applicable improvements talked about in PMFastR [DBZ09](Chapter 2) were also implemented here. These are banding and multithreading. By using banding, we are able to search the genome using large sequences such as 16S and 23S rRNA, it also reduces the execution time considerably by only examining highly likely alignment locations. Multithreading improves wall time, while still keeping the same alignment procedure. This is useful for multi-core environments.

The original underlying assumption for banding is that we are computing a global alignment. Here we apply it to local alignments by this justification: by the use of a scanning window, we reduce the problem to a point where we know the alignment has to exist within certain parameters if it exists at all. We will see later that due to the scanning window concept, we never have a search target that is longer than some pre-defined length. Using this we know that if the query exists in the target, then it must exist with less than the number of gaps consistent with the difference between the lengths We then choose the banding value to be greater than the difference between the profile length and the scanning window size. This allows us to still reduce the running time for longer sequences, but still allows us to search all possible locations of the query in the target.

### 3.2.2 Multiple Alignment Query

The input to the program is a standard ClustalW formatted alignment file, this file contains a portion of the whole genome alignment and an RNA profile. The input the alignment is analyzed and preprocessed before running the alignment procedure. This preprocessing allows the alignment to have high quality even when the actual section of the genome alignment is sparse.

#### 3.2.2.1 Compacting

By compacting the input alignment we are able to remove sections of the input alignment that may be abnormalities. These could be caused by sections that are present in only a few species in the sequence or may be extensions cause by sequencing and assembly errors. A predefined threshold is used to remove any columns that do not contain a certain number of represented species. This means that what is left is a subset of the columns that represent a high quality multiple alignment. This step allows for the discovery of segments that may have been split by one of more events in a low number of species in the alignment. We call this *Horizontal Compacting* as it removes columns from the alignment.

We also implemented *Vertical Compacting*. This is similar in theory and reasoning to Horizontal Compacting but it acts on the other axis. We analyze each sequence to determine the

validity of that sequence in the alignment. We first removed any sequence that is all gaps in this section. This happens a lot in alignments that are very distant on the phylogeny. We then ranked the remaining sequences by the number gaps present. We then chose a specific number of these sequences to be used in the alignment.

For the 15 species Drosophila alignment for example we experimentally chose 5 as the optimal number of sequences. This means only the 5 most represented sequences were used to align the section. That means that in most cases 5-7 of the sequences were removed because they were not present at all in the section and another 5-7 where removed due to their not being dense enough in that particular region. This could be improved to use additional measures to rank the sequence quality.

#### 3.2.2.2 Position Specific Scoring

Because the query is represented by a position specific scoring matrix (PSSM), the score assigned to the column in the target becomes more difficult. The solution presented is a normalized weight of the scores of each of the individual sequences in the alignment. Each sequence contributes a portion to the score, this is done in the same way that the single sequence alignment score is calculated. The sum of these scores is then divided by the number of sequences in the alignment to get the normalized score, this is then used in the dynamic programming to compute the optimal alignment.

The scoring is done simply getting the mean score for each column. The profile of the query allows us to get a score for each letter at each position, so we get the score of each character in the alignment at that column and then normalize this by the number of sequences in the alignment. Equations 3.1 and 3.2 show the definition of the $\gamma$ and $\delta$ functions. Here $i$ and $j$ are the columns in the target and query respectively. The $PSSM_{Single}$ and $PSSM_{Paired}$ arrays are the position specific scoring matrices for the unpaired column $j$ and the paired bases $j_l$ and $j_r$.

$$\gamma(i,j) = \frac{\sum_{s \in Alignment} PSSM_{Single}\left[j\right]\left[s\left(i\right)\right]}{num\_sequences} \tag{3.1}$$

$$\delta(i_l,i_r,j_l,j_r) = \frac{\sum_{s \in Alignment} PSSM_{Paired}\left[j_l\right]\left[j_r\right]\left[s\left(i_l\right)\right]\left[s\left(i_r\right)\right]}{num\_sequences} \tag{3.2}$$

#### 3.2.2.3 Local Alignment

Because the alignment should be local only on the target, very few changes were needed to the alignment procedure. The binarized tree was left as described in Section 1.2.7.1, an optimal alignment was still needed for all nodes. The initialization of the dynamic programming tables was changed such that no penalty was added for insertions at the front of the alignment. Additionally a search was done on the last node of the tree's table to find the

ending location of the alignment, such that no penalty would be given for insertions. To allow for the use of banding, and the ability to align long sequences, the banding value was adjusted to make sure even large gaps at the beginning and end of the input sequence would be searchable. To this end the scanning window, described in Section 3.2.3, could also be adjusted to assure this would be true and allow for low memory consumption.

#### 3.2.2.4 Covarying Mutations

We make special considerations for covarying mutations within the search text. Because we are dealing with a multiple alignment, we want to adjust the scoring to take into account the covarying mutations present within the alignment already at points we predict are structural elements. The covariance score is computed along the way for each alignment combination, $(v, i, j)$, but is not added to the score and does not effect the actual alignment procedure. It does affect the total score of the alignment for comparison with other scores.

The actual method for measuring covariance mutations is somewhat simple. As we mentioned, the covariance score is calculated at each potential pairing location, thus the node $(v)$ and both pairing locations $(i, j)$ are known. We wanted to make sure that we give a bonus to a column that is a mutation from the consensus while still being simple to calculate. The procedure is given in Figure 18. Here the matching bases are defined as the

**procedure covaryScore**$(v, i, j)$
  **for** each $s$ in the multiple alignment
    $count[s_i][s_j]$++
  **end for**
  $score = 0$
  **for** each $(k, l)$ such that $k$ and $l$ are matching bases
      and $(k, l)$ are not equivalent to the bases represented by node $v$
    $score$ += CM_CONST x $\left(1 - \frac{count[k][l]}{num\_sequences}\right)$
  **end for**
  **return** *score*

Figure 18: The Covariance Mutation Score

Watson-Crick bases as well as the wobble base pair. $(k, l)$ is said to be equivalent to the pair in $v$ if $(k, l) = (v_l, v_r)$ or $(k, l) = (v_r, v_l)$.

The covariance mutation (CM) score is then passed along though the alignment procedure similar to that shown in Figure 6(a) just as the array $A$ is passed. This means that say an insert is added to the aligned of $(v, i, j)$, then:

$$A[v][i][j] = A[v][i][j-1] + \gamma('-', j)$$
$$CM[v][i][j] = CM[v][i][j-1]$$

Similarly is the bases are matched then:

$$A[v][i][j] = A[v_{left\_child}][i+1][j-1] + \delta(v_l, v_r, i, j)$$
$$CM[v][i][j] = CM[v_{left_child}][i+1][j-1] + covaryScore(v, i, j)$$

### 3.2.3 Scanning

To scan an entire genome, or chromosome, for a single query a scanning procedure is implemented to break the long alignment into workable pieces. A scanning window was defined to allow for a certain length of the alignment to be examined at a time. Each segment of the length of the scanning window was passed to the alignment procedure to see if a high quality alignment could be found. An increase in the size of the window would allow for a decrease in the number of iterations of the alignment procedure, but an increase in the memory consumption. Additionally, an increment length is also defined, this is the distance from one scanning window to the next along the target alignment. A decrease in the increment would allow for a more precise searching of the genome, but would require more iterations of the alignment procedure, also more repeated hits would be found.

Figure 19 shows the scanning window. It can be seen that if the window size is increased more of the genome is covered with each iteration of the alignment procedure, but the accuracy to find multiple hits is diminished. Similarly if the increment size is increased, fewer iterations again are needed but you have a higher probability to miss hits that span two windows in the search.

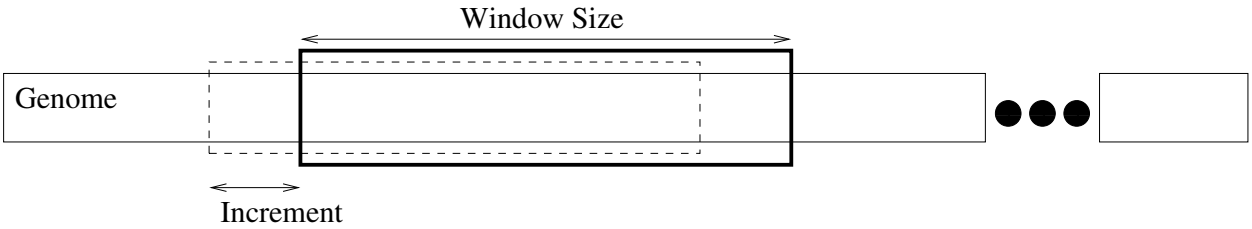



Figure 19: Scanning Window.

## 3.3 CONCLUSION

What we have done in this chapter is propose a new procedure for aligning alignments. Though many people have attempted and proposed solutions to this problem, we are the first to do so using RNA structure. This is useful for searching the newly available multiple species genome alignments, and it should provide more information into the actual meaning behind the sequences beyond what can be achieved by just a single sequence search.

# CHAPTER 4:
# CONCLUSION

Here we present two algorithms that are influential in the area of computational biology. Both use known unique elements in ncRNA to help guide alignments. These alignments are used in two new ways: creating multiple alignment profiles and searching genome alignments. Not only are these algorithms unique in their solution, they are also efficient enough to be used on very large known sequences such as 16S and 23S rRNA.

PMFastR uses these ideas to construct multiple alignments not only on par with other state of the art multiple alignment programs, but we show that it can create multiple alignments of similar quality to those produced by hand curation.

AlnAlign takes the idea of sequence-structure alignment one step further and applies this idea to search; not only search but multiple alignment search. Using the information from multiple species you should be able to gather more information than from a single sequence.

## 4.1 FUTURE WORK

We can see that while the algorithms presented here make a point in the community they both have areas where they can be improved. The first step would be to remove the CMBuild refinement step from PMFastR, if this implementation could be recreated, not only would the refinement be able to be performed at each step but could be run only on the portions of the profile unaligned by the PMFastR algorithm.

In addition to improvements in both programs, all of the major ideas presented here can be brought into other areas of computational biology. Banding can be implemented in any alignment, not just structure based ones. Many examples can already be found on how progressive multiple alignments are used elsewhere. The ideas presented about multithreading could be applied to almost any tree based search or computation algorithm. As we become a computer science community where multi-cored machines are more prevalent, the transition to making all of our programs utilize this environment should be explored everywhere.

Going backwards, the algorithm presented in the conception of AlnAlign can also be applied to the multiple alignment creation of PMFastR. While many multiple alignment programs follow a tree structure to guide the alignment. In doing so they have to align multiple alignment. We avoid this step by only aligning one sequence to the profile at a time. The AlnAlign procedure may be able to be used to improve the results of PMFastR beyond what results we already have.

Sequence weighting can be applied to the AlnAlign procedure. For instance in the 15 species insect genome alignment, we assume that all sequences would have the same impact on the alignment. If the actual seed were weighted higher, contributes more to the score calculation, this would force the algorithm to align more strongly with the seed sequence. This could be equivalent to aligning to the seed sequence, given the rest of the alignment as a background influence.

# LIST OF REFERENCES


[ABF94] Amihood Amir, Gary Benson, and Martin Farach. "Let sleeping files lie: pattern matching in Z-compressed files." In *SODA '94: Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pp. 705–714, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[AC75] Alfred V. Aho and Margaret J. Corasick. "Efficient string matching: an aid to bibliographic search." *Commun. ACM*, **18**(6):333–340, 1975.

[AGM90] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. "Basic local alignment search tool." *J. Mol. Biol.*, **215**:403–410, Oct 1990.

[Aho90] Alfred V. Aho. "Algorithms for finding patterns in strings." pp. 255–300, 1990.

[AMS97] S. F. Altschul, T. L. Madden, A. A. Schffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs." *Nucleic Acids Res.*, **25**:3389–3402, Sep 1997.

[AN09] A. Amir and G. Navarro. "Parameterized Matching on Non-linear Structures." *Information Processing Letters*, 2009. To appear.

[Bai92] A. Bairoch. "PROSITE: a dictionary of sites and patterns in proteins." *Nucleic Acids Res.*, **20 Suppl**:2013–2018, May 1992.

[Bai93] A. Bairoch. "The PROSITE dictionary of sites and patterns in proteins, its current status." *Nucleic Acids Res.*, **21**:3097–3103, Jul 1993.

[Bak78] Theodore P. Baker. "A Technique for Extending Rapid Exact-Match String Matching to Arrays of More than One Dimension." *SIAM Journal on Computing*, **7**(4):533–541, 1978.

[Bir77] "Two Dimensional Pattern Matching." *Information Processing Lett.*, **6**:168–170, 1977.

[BKR07] Markus Bauer, Gunnar Klau, and Knut Reinert. "Accurate multiple sequence-structure alignment of RNA sequences using combinatorial optimization." *BMC Bioinformatics*, **8**(1):271, 2007.

[BM77] Robert S. Boyer and Strother J. Moore. "A Fast String Searching Algorithm." *Communications of the ACM*, **20**(10):762–772, October 1977.

[BMR95] V. Bafna, S. Muthukrishnan, and R. Ravi. "Computing similarity between RNA strings." In Z. Galil and E. Ukkonen, editors, *Proceedings of the6th Annual Symposium on Combinatorial Pattern Matching*, pp. 1–16, Espoo, Finland, 1995. Springer-Verlag, Berlin.

[Brz62] J A Brzozowski. "A survey of regular expressions and their applications." *IRE transactions on Electronic Computers*, (11):324–335, 1962.

[Brz64] Janusz A. Brzozowski. "Derivatives of Regular Expressions." *J. ACM*, **11**(4):481–494, 1964.

[BSS74] William A. Beyer, Myron L. Stein, Temple F. Smith, and Stanislaw M. Ulam. "A molecular sequence metric and evolutionary trees." *Mathematical Biosciences*, **19**(1-2):9 – 25, 1974.

[BWF00] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. "The Protein Data Bank." *Nucleic Acids Res.*, **28**:235–242, Jan 2000.

[CL88] Humberto Carrillo and David Lipman. "The multiple sequence alignment problem in biology." *SIAM J. Appl. Math.*, **48**(5):1073–1082, 1988.

[CN09] F. Claude and G. Navarro. "Self-Indexed Text Compression using Straight-Line Programs." In *Proc. 34th International Symposium on Mathematical Foundations of Computer Science(MFCS)*, LNCS. Springer, 2009. To appear.

[CSS02] Jamie Cannone, Sankar Subramanian, Murray Schnare, James Collett, Lisa D'Souza, Yushi Du, Brian Feng, Nan Lin, Lakshmi Madabusi, Kirsten Muller, Nupur Pande, Zhidi Shang, Nan Yu, and Robin Gutell. "The Comparative RNA Web (CRW) Site: an online database of comparative sequence and structure information for ribosomal, intron, and other RNAs." *BMC Bioinformatics*, **3**(1):2, 2002.

[DBZ09] Daniel F DeBlasio, Jocelyn Braund, and Shaojie Zhang. "PMFastR: A New Approach to Multiple RNA Structure Alignment." In *The Workshop on Algorithms in Bioinformatics (WABI)*, Sep. 2009.

[DMB05] C. B. Do, M. S. Mahabhashyam, M. Brudno, and S. Batzoglou. "ProbCons: Probabilistic consistency-based multiple sequence alignment." *Genome Res.*, **15**:330–340, Feb 2005.

[DNZ98] E.S. De Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. "Direct pattern matching on compressed text." In *String Processing and Information Retrieval: A South American Symposium*, pp. 90–95, Sep 1998.

[DWB06] C. B. Do, D. A. Woods, and S. Batzoglou. "CONTRAfold: RNA secondary structure prediction without physics-based models." *Bioinformatics*, **22**:e90–98, Jul 2006.

[DWM06] 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*, **22**:1593–1599, Jul 2006.

[ED94] S. R. Eddy and R. Durbin. "RNA sequence analysis using covariance models." *Nucleic Acids Res.*, **22**:2079–2088, Jun 1994.

[Edd] S. R. Eddy. "Infernal package http://infernal.janelia.org/.".

[Edg04] R.C. Edgar. "MUSCLE: multiple sequence alignment with high accuracy and high throughput." *Nucleic Acids Res.*, **32**:1792–1797, 2004.

[FGN09] P. Ferragina, R. González, G. Navarro, and R. Venturini. "Compressed Text Indexes: From Theory to Practice." *ACM Journal of Experimental Algorithmics (JEA)*, **13**:article 12, 2009. 30 pages.

[FP74] M. J. Fischer and M. S. Paterson. "STRING-MATCHING AND OTHER PRODUCTS." Technical report, Cambridge, MA, USA, 1974.

[Gal79] Zvi Galil. "On improving the worst case running time of the Boyer-Moore string matching algorithm." *Commun. ACM*, **22**(9):505–508, 1979.

[GBM03] Sam Griffiths-Jones, Alex Bateman, Mhairi Marshall, Ajay Khanna, and Sean R. Eddy. "Rfam: an RNA family database." *Nucleic Acids Res.*, **31**(1):439–441, 2003.

[GKS95] S. K. Gupta, J. D. Kececioglu, and A. A. Schffer. "Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment." *J. Comput. Biol.*, **2**:459–472, 1995.

[GMM05] S. Griffiths-Jones, S. Moxon, M. Marshall, A. Khanna, S. R. Eddy, and A. Bateman. "Rfam: annotating non-coding RNAs in complete genomes." *Nucleic Acids Res.*, **33**:D121–124, Jan 2005.

[Gus97] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, January 1997.

[GWW05] P. P. Gardner, A. Wilm, and S. Washietl. "A benchmark of multiple sequence alignment programs upon structural RNAs." *Nucleic Acids Res.*, **33**:2433–2439, 2005.

[HBS04] I. L. Hofacker, S. H. Bernhart, and P. F. Stadler. "Alignment of RNA base pairing probability matrices." *Bioinformatics*, **20**:2222–2227, Sep 2004.

[HFS02] I. L. Hofacker, M. Fekete, and P. F. Stadler. "Secondary structure prediction for aligned RNA sequences." *J. Mol. Biol.*, **319**:1059–1066, Jun 2002.

[Hol05] I. Holmes. "Accelerated probabilistic inference of RNA structure evolution." *BMC Bioinformatics*, **6**:73, 2005.

[HST06] Eli Hershkovitz, Guillermo Sapiro, Allen Tannenbaum, and Loren Dean Williams. "Statistical Analysis of RNA Backbone." *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, **3**(1):33, 2006.

[JLM02] Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. "A general edit distance between RNA structures." *Journal of Computational Biology*, **9**:2002, 2002.

[Joh79] Steven C. Johnson. "Yacc: Yet Another Compiler Compiler." Technical report, New York, NY, USA, 1979.

[JTZ89] J. A. Jaeger, D. H. Turner, and M. Zuker. "Improved predictions of secondary structures for RNA." *Proc. Natl. Acad. Sci. U.S.A.*, **86**:7706–7710, Oct 1989.

[KE03] R. J. Klein and S. R. Eddy. "RSEARCH: finding homologs of single structured RNA sequences." *BMC Bioinformatics*, **4**:44, Sep 2003.

[Kec93] John D. Kececioglu. "The Maximum Weight Trace Problem in Multiple Sequence Alignment." In *CPM '93: Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching*, pp. 106–119, London, UK, 1993. Springer-Verlag.

[KH03] B. Knudsen and J. Hein. "Pfold: RNA secondary structure prediction using stochastic context-free grammars." *Nucleic Acids Res.*, **31**:3423–3428, Jul 2003.

[KJP77] Donald E. Knuth, Jr, and Vaughan R. Pratt. "Fast Pattern Matching in Strings." *SIAM Journal on Computing*, **6**(2):323–350, 1977.

[KKT05] K. Katoh, K. Kuma, H. Toh, and T. Miyata. "MAFFT version 5: improvement in accuracy of multiple sequence alignment." *Nucleic Acids Res.*, **33**:511–518, 2005.

[KL05] Y. Karklin and M. S. Lewicki. "A hierarchical Bayesian model for learning nonlinear statistical regularities in nonstationary natural signals." *Neural Comput*, **17**:397–423, Feb 2005.

[Kle56] S. Kleene. *Automata Studies*. Princeton, N.J., 1956.

[KLS92] L. F. Kolakowski, J. A. Leunissen, and J. E. Smith. "ProSearch: fast searching of protein sequences with regular expression patterns related to protein structure and function." *BioTechniques*, **13**:919–921, Dec 1992.

[KS04] John Kececioglu and Dean Starrett. "Aligning alignments exactly." In *RECOMB '04: Proceedings of the eighth annual international conference on Resaerch in computational molecular biology*, pp. 85–96, New York, NY, USA, 2004. ACM.

[KTH02] P. S. Klosterman, M. Tamura, S. R. Holbrook, and S. E. Brenner. "SCOR: a Structural Classification of RNA database." *Nucleic Acids Res.*, **30**:392–394, Jan 2002.

[KTK07] Hisanori Kiryu, Yasuo Tabei, Taishin Kin, and Kiyoshi Asai. "Murlet: a practical multiple alignment tool for structural RNA sequences." *Bioinformatics*, **23**(13):1588–1598, 2007.

[LAK89] D.J. Lipman, S.F. Altschul, and J.D. Kececioglu. "A tool for multiple sequence alignment." *Proc. Natl. Acad. Sci. U.S.A.*, **86**:4412–4415, Jun 1989.

[LBB07] M.A. Larkin, G. Blackshields, N.P. Brown, R. Chenna, P.A. McGettigan, H. McWilliam, F. Valentin, I.M. Wallace, A. Wilm, R. Lopez, J.D. Thompson, T.J. Gibson, and D.G. Higgins. "Clustal W and Clustal X version 2.0." *Bioinformatics*, **23**(21):2947–2948, 2007.

[LC98] Anna J Lee and Donald M Crothers. "The solution structure of an RNA loop-loop complex: the ColE1 inverted loop sequence." *Structure*, **6**(8):993 – 1007, 1998.

[Lev66] Vladimir I. Levenshtein. "Binary codes capable of correcting deletions, insertions, and reversals." *Soviet Physics Doklady*, **10**(8):707–710, 1966.

[Mai78] David Maier. "The Complexity of Some Problems on Subsequences and Supersequences." *J. ACM*, **25**(2):322–336, 1978.

[MBH95] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. "SCOP: a structural classification of proteins database for the investigation of sequences and structures." *J. Mol. Biol.*, **247**:536–540, Apr 1995.

[McC76] Edward M. McCreight. "A Space-Economical Suffix Tree Construction Algorithm." *J. ACM*, **23**(2):262–272, 1976.

[MM90] Udi Manber and Gene Myers. "Suffix Arrays: A New Method for On-Line String Searches." In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pp. 319–327. Society for Industrial and Applied Mathematics, 1990.

[MT02] D. H. Mathews and D. H. Turner. "Dynalign: an algorithm for finding the secondary structure common to two RNA sequences." *J. Mol. Biol.*, **317**:191–203, Mar 2002.

[MY60] R McNaughton and H Yamada. "Regular expressions and state graphs for automats." *IRE transactions on Electronic Computers*, pp. 39–47, 1960.

[NHH00] C. Notredame, D. G. Higgins, and J. Heringa. "T-Coffee: A novel method for fast and accurate multiple sequence alignment." *J. Mol. Biol.*, **302**:205–217, Sep 2000.

[RE01] E. Rivas and S. R. Eddy. "Noncoding RNA gene detection using comparative sequence analysis." *BMC Bioinformatics*, **2**:8, 2001.

[San85] D. Sankoff. "Simulations solution of the RNA folding, alignment and protosequence problems." *SIAM J. Appl. Math.*, **45**(5):810–825, 1985.

[SB05] S. Siebert and R. Backofen. "MARNA: multiple alignment and consensus structure prediction of RNAs based on sequence structure comparisons." *Bioinformatics*, **21**:3352–3359, Aug 2005.

[SBB02] Guenter Stoesser, Wendy Baker, Alexandra van den Broek, Evelyn Camon, Maria Garcia-Pastor, Carola Kanz, Tamara Kulikova, Rasko Leinonen, Quan Lin, Vincent Lombard, Rodrigo Lopez, Nicole Redaschi, Peter Stoehr, Mary Ann Tuli, Katerina Tzouvara, and Robert Vaughan. "The EMBL Nucleotide Sequence Database." *Nucleic Acids Res.*, **30**(1):21–26, 2002.

[SK83] David Sankoff and Joseph B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley Publishing Co, Reading, Massachusetts, 1983.

[Sta91] R Staden. "Screening protein and nucleic acid sequences against libraries of patterns." *DNA Sequence*, (1):1–369, 1991.

[Ste91] Michael J.E. Sternberg. "PROMOT: a FORTRAN program to scan protein sequences against a library of known motifs." *Comput. Appl. Biosci.*, **7**(2):257–260, 1991.

[THG94] 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." *Nucleic Acids Res.*, **22**:4673–4680, Nov 1994.

[THG07] E. Torarinsson, J. H. Havgaard, and J. Gorodkin. "Multiple structural alignment and clustering of RNA sequences." *Bioinformatics*, **23**:926–932, Apr 2007.

[THK04] M. Tamura, D. K. Hendrix, P. S. Klosterman, N. R. Schimmelman, S. E. Brenner, and S. R. Holbrook. "SCOR: Structural Classification of RNA, version 2.0." *Nucleic Acids Res.*, **32**:D182–184, Jan 2004.

[Tho68] Ken Thompson. "Programming Techniques: Regular expression search algorithm." *Commun. ACM*, **11**(6):419–422, 1968.

[TPP99] J. D. Thompson, F. Plewniak, and O. Poch. "A comprehensive comparison of multiple sequence alignment programs." *Nucleic Acids Res.*, **27**:2682–2690, Jul 1999.

[Ukk95] Esko Ukkonen. "On-Line Construction of Suffix Trees." *Algorithmica*, **14**(3):249–260, 1995.

[War95] H. T. Wareham. "A simplified proof of the NP- and MAX SNP-hardness of multiple sequence tree alignment." *J. Comput. Biol.*, **2**:509–514, 1995.

[Wei73] Peter Weiner. "Linear pattern matching algorithms." In *SWAT '73: Proceedings of the 14th Annual Symposium on Switching and Automata Theory (swat 1973)*, pp. 1–11, Washington, DC, USA, 1973. IEEE Computer Society.

[WHL05] S. Washietl, I. L. Hofacker, M. Lukasser, A. Httenhofer, and P. F. Stadler. "Mapping of conserved RNA secondary structures predicts thousands of functional noncoding RNAs in the human genome." *Nat. Biotechnol.*, **23**:1383–1390, Nov 2005.

[WJ94] L. Wang and T. Jiang. "On the complexity of multiple sequence alignment." *J Comput Biol*, **1**(4):337–348, 1994.

[WMS06] A. Wilm, I. Mainz, and G. Steger. "An enhanced RNA alignment benchmark for sequence alignment programs." *Algorithms Mol Biol*, **1**:19, 2006.

[WR04] Zasha Weinberg and Walter L. Ruzzo. "Exploiting conserved structure for faster annotation of non-coding RNAs without loss of accuracy." *Bioinformatics*, **20**(suppl_1):i334–341, 2004.

[ZBA06] Shaojie Zhang, Ilya Borovok, Yair Aharonowitz, Roded Sharan, and Vineet Bafna. "A sequence-based filtering method for ncRNA identification and its application to searching for riboswitch elements." *Bioinformatics*, **22**(14):e557–e565, 2006.

[ZHE05] Shaojie Zhang, B. Haas, E. Eskin, and V. Bafna. "Searching genomes for noncoding RNA using FastR." *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, **2**(4):366–379, Oct.-Dec. 2005.

[ZS84] M. Zuker and D. Sankoff. "RNA secondary structures and their prediction." *Bulletin of Mathematical Biology*, **46**(4):591–621, 1984.