Accurate annotation of non-coding RNAs in practical time

Zasha Weinberg

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

2005

Program Authorized to Offer Degree: Computer Science and Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Zasha Weinberg

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of the Supervisory Committee:

Walter L. Ruzzo

Reading Committee:

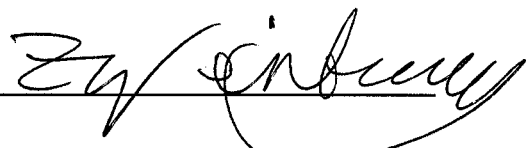
Walter L. Ruzzo

Martin Tompa

James H. Thomas

Date: August 18, 2005

In presenting this dissertation in partial fulfillment of the requirements for the doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Proquest Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature 

Date Aug 18, '05

University of Washington

Abstract

Accurate annotation of non-coding RNAs in practical time

Zasha Weinberg

Chair of the Supervisory Committee:
Professor Walter L. Ruzzo
Computer Science and Engineering

Several times each year, one thousand computers at the Sanger Institute in England spend two weeks updating the Rfam Database.

This intensive effort is needed to support recent surprising discoveries showing that RNAs are much more powerful and biologically significant than previously realized, discoveries that upset decades of assumptions in molecular biology.

The Rfam Database is a collection of functional RNAs not coding for proteins, the so-called non-coding RNAs (ncRNAs). The Rfam Database groups ncRNAs into evolutionarily related families, and searches 8 billion nucleotides of genome sequences for new ncRNAs that are members of these families.

This search is done with a Covariance Model (CM), a statistical model based on probabilistic context-free grammars. Although CMs have excellent accuracy, they are infeasibly slow: a pure CM-based implementation of the Rfam Database would require 10,000 CPU years. So, Rfam uses an *ad hoc* heuristic (based on BLAST, a popular program not specialized for RNAs) to reduce this time to 2,000 CPU weeks—at an unknown cost to sensitivity.

This dissertation work significantly improves on CMs by designing CM-based algorithms that are roughly one hundred times faster, yet preserve all or most of the CM's sensitivity. All of these algorithms filter sequences, eliminating unpromising sub-sequences, and running the slow CM only on the most promising sub-sequences.

One class of filter, the *rigorous filter*, guarantees that it will never eliminate subsequences that the CM would recognize as an ncRNA. In other words, the use of a rigorous filter cannot compromise sensitivity. Such a filter is unusual in computational biology, where filters typically make no guarantees at all. Our basic rigorous filters use probabilistic regular grammars, with linear inequalities on rule scores guaranteeing rigorousness. More powerful classes of filter exploit limited secondary structure to gain discriminative power without unduly compromising speed.

We further develop a class of *heuristic filters* that scan faster, at a modest cost to sensitivity—a desirable trade-off in many contexts. This dissertation empirically measures speed and sensitivity of heuristic filters on real biological data, providing an objective analysis of various filters' actual performance.

These techniques allow Rfam Database searches in roughly the same time as the current solution, but yield new ncRNAs missed by the *ad hoc* filters that were necessary for practical CM searches until now. These techniques were applied in collaboration with experimental biologists. Among other contributions, these searches (1) led to the first discovery of a naturally occurring RNA (a glycine-binding "riboswitch") that uses cooperative binding, a sophisticated biochemical mechanism previously known only in proteins, and (2) assisted in finding 6S RNA in virtually all groups of bacteria, whereas 6S had been known only in the γ-proteobacteria group for roughly 30 years.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

Annotation: Genomes or sequence databases are said to be annotated with ncRNAs. In the context of this dissertation this means that the nucleotide locations of ncRNAs in the sequence database are determined, and it is specified to which ncRNA family the annotated ncRNA is homologous.

Archaea: Archaea are single-celled life forms that lack a distinct nuclear compartment. They are considered distinct from the bacteria, mainly based on their evolutionary divergence, but in the context of this dissertation the distinction is not important.

Bacteria: Bacteria are single-celled life forms that lack a distinct nuclear compartment.

Base: In the context of this dissertation, roughly a synonym of "nucleotide".

Base pair: See chapter 1.

BLAST: A computer algorithm and implementation [1]. Given as input a query DNA sequence and database, BLAST will search the database for approximate matches to the query. The algorithm is unaware of RNA secondary structure.

Bulge: An RNA secondary structure pattern. A bulge is one or more consecutive single-stranded nucleotides connecting two helices. See chapter 1.

CFG: See "SCFG".

cis: A *cis* element is a subsequence that is linearly close to another subsequence in the same DNA molecule, where the association is functionally important. For example,

a riboswitch needs to be a part of an mRNA to regulate that mRNA, so it is a *cis* element.

Context-free grammar: See "SCFG".

CM: abbreviation of "Covariance model" (q.v.).

Compensatory mutation: A compensatory mutation reverses the deleterious effect of some previous mutation. For example, suppose some A-U base pair is vital to the correct RNA secondary structure of some RNA. An A→C mutation would result in a C-U base "pair", which would compromise the stability of the desired secondary structure. However, a subsequent U→G mutation would result in a C-G base pair, compensating for the first mutation.

Complementary: Two RNA (or DNA) sequences are complementary if they can easily bind each other. (Technically, they are "reverse complements".) Usually this means that when lined in, their nucleotides can base pair.

Conserved/conservation: See section 1.1.6.

Covariance model: A type of model that describes an ncRNA family, and is used to find homologs. See chapter 2 and chapter 3.

DNA: See chapter 1.

Eubacteria: A synonym of bacteria. (The term "eubacteria" makes a clearer distinction that it is not archaeal, which are sometimes called archaebacteria).

Eukaryote: The eukaryotes are one of the three main domains of life (the others being archaea and bacteria). Eukaryotes include all plants and animals, thus I expect that all readers of this dissertation are eukaryotes. Many single-celled organisms are eukaryotes. Unlike archaea and bacteria, a eukaryote has a nucleus within its cell(s).

Family: In the context of this dissertation, an ncRNA *family* is a set of ncRNAs that are evolutionarily related, i.e., homologous. In other words, they share the same biological function (or at least a similar one), and are presumed to have evolved from some single, original RNA sequence.

Filtering fraction: A metric for evaluating the selectivity of a sequence filter. Filtering fraction is the amount of the database sequence left after filtering; the slower CM must be run on this remaining sequence. Thus, a fraction of 0 implies the fastest possible scans (since the CM is never run at all), but cannot hope to find any homologs (also since the CM is never run). A filtering fraction of 1 is pointless; the scan is the same as a raw scan, and somewhat slower since the filter consumes additional CPU time. Filtering fraction is used in most technical chapters of this dissertation, and defined in each one. The first introduction of the concept is in section 4.1.

Fraction: see "Filtering fraction".

G+C content: The fraction of nucleotides that are either G or C, usually expressed as a percentage.

Gene: See chapter 1.

Genome: The complete set of DNA sequences in an organism. For example, the human genome consists of 25 non-redundant DNA molecules called chromosomes (22 non-sex chromosomes, the X and Y chromosomes and the mitochondrial chromosome, often itself called the mitochondrial genome).

Grammar: See "SCFG" and "HMM". More information on formal grammars is in [27].

Gigabase: one billion nucleotides.

Hit: In this dissertation, a "hit" is any subsequence that a CM predicts is an ncRNA family member. Some hits are true homologs, while other hits are false positives.

HMM: Hidden Markov Model, a probabilistic model used for biological sequences. HMMs can be viewed as a kind of probabilistic finite automaton, and are equivalent in power to probabilistic regular grammars.

Hairpin: RNA structure pattern that has a helix and a terminal loop. In this dissertation, I use the most general definition of a hairpin, allowing it to include any internal loops or bulges. Many authors restrict hairpins to only one helix with no internal loops/bulges, but some usage allows many (e.g., pre-microRNAs are called hairpins). See section 1.1.5.

Heuristic: A heuristic is an algorithm that attempts to perform well, but makes no guarantees about its performance. Heuristic filters in this dissertation have sound reasons suggesting that they will work well, but cannot guarantee sensitivity.

Hit: When a CM predicts an ncRNA homolog at some location, I call that a "hit".

Homolog: Two RNAs are homologous if they are similar because they share a common ancestry, e.g., an RNA present in humans and chimps that was originally in the ancestral species of human and chimp. See chapter 1.

Indel: Short for "INsertion or DELetion". In the context of this dissertation, an indel in a homolog is a place where there is an extra or missing nucleotide relative to the expected conserved nucleotide pattern defined by a Multiple Sequence Alignment.

Infernal: Infernal is a software package created by Sean Eddy that implements algorithms to create CMs and scan genome sequences with them.

Infinite-length forward algorithm: A method to train rigorous profile HMM filters. See chapter 4.

Kilobase: One thousand nucleotides.

Local CM: A variant of the CM that allows large deletions or insertions at a fixed cost. This can, for example, allow the discovery of ncRNAs with a large part missing, like tRNAs missing one arm of their classic cloverleaf structure. Local CMs are related to local alignments, e.g., in the Smith-Waterman algorithm. See chapter 3.

LOD score: A score derived from the Logarithm of the ODds of two competing probabilistic models.

Megabase: One million nucleotides.

Millibase: One thousandth of a nucleotide. Just kidding.

ML-heuristic: A heuristic filter for CMs that is part of this dissertation work. See chapter 7 and chapter 8.

mRNA: messenger RNA. See section 1.1.3.

MSA: See Multiple Sequence Alignment.

Multiple Sequence Alignment: Given the sequences of multiple RNAs (or DNAs or proteins), a Multiple Sequence Alignment (MSA) indicates how nucleotide positions relate to each other, by aligning related positions so that they fall into the same column. For RNA, MSAs typically indicate which columns base pair with each other. See Figure 1.2 for a tiny, hypothetical example. Chapter 10 includes some real MSAs.

Mutation: DNA nucleotides can mutate (i.e., change) usually due to environmental factors like ultraviolet radiation and certain chemicals, or due to errors when DNA is

copied in cell division. If mutated DNA is used to make RNAs or proteins, they may also be mutated. Most mutations are bad, but some mutations are benign or even beneficial.

ncRNA: Abbreviation of non-coding RNA. I define any RNA with a biological function that does not involve coding for proteins to be an ncRNA. This definition includes regulatory elements in mRNAs and independently transcribed ncRNA genes. See chapter 1.

Non-coding RNA: See ncRNA.

Nucleotide: See chapter 1.

Phylogeny: Phylogeny refers to the evolutionary history of organisms or genome sequences (and the RNAs and proteins that the genome creates). Hence if organisms are said to be "phylogenetically close", those organisms are closely related and have a relatively recent common ancestor. Humans and chimps are commonly considered to be phylogenetically close.

Profile HMM: A profile HMM is a type of statistical model used for families of related DNA or protein sequences [27]. Profile HMMs are used in this dissertation as filters. See chapter 2.

Prokaryote: The prokaryotes include all archaea and bacteria.

Protein: See chapter 1.

Pseudoknot: An RNA molecule has a pseudoknot if has base pair interactions that cross one another. In other words, if base pairs form between nucleotides i and j, and between k and l, with $i < k < j$ and $k < j < l$. Pseudoknot structures cannot, in general, be represented by context-free grammars.

RaveNnA: RAVENNA [129] is a software package that I created that implements the algorithms described herein. Ravenna is a also neighborhood of Seattle in which I lived while at the University of Washington.

Regular grammar: A type of formal grammar. Probabilistic versions of regular grammars are equivalent in power to HMMs. More information on formal grammars is in [27].

Rfam Database: A database of ncRNA families [48]. The current release (March 2005) has 503 families. Each family is based on a hand-curated multiple alignment that is used to create a CM. The CM is then used to search for additional family members.

RFAMSEQ: A genome sequence database that is a subset of the EMBL nucleotide database [118]. The Rfam Database (q.v.) uses its CMs to annotate known ncRNAs within RFAMSEQ. As of 2005, RFAMSEQ is roughly 8 Gigabases.

Rigorous: In the context of this dissertation, a "rigorous filter" is one guaranteeing that it never eliminates any homolog that the CM would recognize. A "rigorous scan" is a scan of a genome database using a rigorous filter.

Riboswitch: A riboswitch is a regulatory RNA. In particular, it is a part of an mRNA that directly regulates its own activity in response to the presence or absence to a specific small molecule (e.g., vitamin B_{12} molecules).

RNA: See chapter 1.

ROC-like curve: See section 7.1.1.

RSEARCH: A computer program. Given a single query RNA sequence, RSEARCH will find approximate matches in the database, and uses CMs to do this. [61].

rRNA: Ribosomal RNA, a class of RNA. See [46].

SCFG: Stochastic context-free grammar, a model used for RNAs that is based on probabilistic formal grammars. See section 1.2 for more details on SCFGs in general, or chapter 3 for how SCFGs are used for covariance models.

Secondary structure: In this dissertation, secondary structure refers to the set of base pairs formed among nucleotides in one RNA molecule. The techniques in this dissertation disallow pseudoknots, structures with crossing interactions that cannot be modeled in context-free grammars.

Sensitivity: In the context of ncRNA homology search, sensitivity answers the question: of the true homologs in the sequence database, what fraction can the search discover? More formally, given a prediction algorithm, sensitivity is TP/(TP+FN), where TP is the number of True Positives (positive predictions that the algorithm makes that are correct) and FN is the number of False Negatives (negative predictions where the correct answer was positive).

Sequence: Used in two contexts: (1) a linear sequence of DNA or RNA nucleotides, (2) information within such sequences that do not reflect the distant correlations between base pairs induced by conserved secondary structure. See chapter 1.

Single-stranded: RNA nucleotides that are not involved in base pairs. See chapter 1.

Specificity: In the context of ncRNA homology searches, specificity answers the question: how many false positives will the search find relative to the sequence database size? More formally, given a prediction algorithm, FN/(FN+FP) where FN is the number of False Negatives (negative predictions where the correct answer was positive) and FP is the number of False Positives (positive predictions where the correct answer was negative). If we define specificity on a nucleotide level, then 1 minus specificity is the number of false positives per nucleotide unit.

Statistical profile: Statistical profiles are a class of techniques to find homologs of families. In the context of ncRNA families, statistic profiles are covariance models and ERPIN. For proteins and DNA sequences, profile HMMs are the most common statistic profile technique.

Stochastic Context-Free Grammar: see SCFG.

Structure: In the context of this dissertation, refers to secondary structure.

tRNA: tRNA (transfer RNA) is one type of ncRNA [46].

tRNAscan-SE: tRNAscan-SE is a program to annotate tRNAs, i.e., discover them in sequence databases or genomes [82]. This program is highly successful in the sense that it has been shown to have superior sensitivity and specificity, and in that it is used in virtually all genome projects. tRNAscan-SE uses CMs. As heuristic filters, it leverages two previously made tRNA detection programs.

Window length: In the standard covariance model framework, a parameter must be set by the user defining the maximum length (in nucleotides) that a member of a particular family can be. This parameter is the *window length*. The window length is a factor in the run time complexity of CM scans.

ACKNOWLEDGMENTS

I was supported by National Institutes of Health grants R01 HG02602 and NIH HG-00035 and National Science Foundation grant NSF DBI-9974498. I extensively used computers donated by IBM. I am particularly grateful for financial support from Martin Tompa, who asked nothing in return.

My work on non-local rigorous filters is published [130, 131]. Two papers on heuristic filters and local filters are under submission. Biological discoveries related to this dissertation work are published [86, 6].

I would like to thank my committee Martin Tompa, Jim Thomas, Raj Rao, Tim Rose, Jens-Erik Mai and particularly my advisor Larry Ruzzo. If I have seen far, it is because I have sat on the chair of my committee.

The science presented in this dissertation greatly benefited from interactions with Alex Bateman, Amol Prakash, Bill Noble, Bob Waterston, Dan Grossman (the younger), David Syphers, Diana Kolbe, Divya Bhat, Jeff Barrick, Kasia Wilamowska, Kathy Collins, Michal Linial, Nan Li, Phil Green, Ron Breaker, Sam Griffiths-Jones, Sean Eddy, Simon Moxon, Todd Lowe, Zizhen Yao and my committee members.

Two people have indirectly but closely shaped my Ph.D.: Jim Kennedy and Dennis Shasha. I worked as a summer student in 1994 in Jim Kennedy's psychiatric neurogenetics lab at the Clarke Institute for Psychiatry. Jim and his lab introduced me to genetics and to computational biology. Dennis Shasha, a CS professor at NYU, was a mentor for me while an undergrad and in the time between my deciding to do a Ph.D. and starting at U.W.

There are moreover a great many others to thank for direct and indirect help on my research, my career and my life. Thanks! In the interests of keeping this dissertation to its already gigantic size (but no larger), I'll leave it at thanking my parents, whose limited understanding of biology was enough to give me a start.

DEDICATION

This dissertation is dedicated to itself, without which it could never have existed.

Chapter 1

BACKGROUND IN COMPUTATIONAL BIOLOGY

This chapter gives background for two audiences:

- Biology for computer scientists who are unfamiliar with genetics.
- Context-free grammars for non-computer scientists.

1.1 Biology for computer scientists

1.1.1 DNA and its sequence

Genetic information is stored in DNA molecules. A reasonable abstraction is that DNA is a very long chain of small units called *nucleotides* or *bases*. There are four types of nucleotides in DNA, which are abbreviated A, C, G and T. A DNA sequence specifies the sequence of nucleotides used in a DNA molecule, i.e., it is a string over the alphabet {A,C,G,T}.

1.1.2 RNA

RNA molecules resemble DNA. For this dissertation, the major difference is that instead of DNA's T nucleotide, RNA uses U; thus, RNA is a string over {A,C,G,U}.

1.1.3 Genes are subsequences; they make proteins via an intermediary RNA

Some subsequences within DNA sequences are called *genes*, and have important functions. For the most part, the function of a gene is to create a *protein* molecule. Proteins are complex molecules that carry out many tasks of importance to the cell, e.g., letting specific nutrients into the cell or catalyzing some chemical reaction that is part of metabolism.

The first step in creating a gene's protein is to copy the DNA sequence of the gene into an RNA molecule. The RNA's sequence is identical to the DNA sequence except that each nucleotide T is replaced by U.

The next step is to use this RNA molecule as a template from which to create the protein molecule. This process is unimportant for my dissertation. (I have also not described exceptions and complications to the above process, but this simplified version is sufficient for the dissertation.) The RNA molecules that are used to create proteins are called *messenger RNAs*, or *mRNAs*.

1.1.4 RNAs can play important roles

This dissertation is about RNAs that are more powerful than being mere passive messages. These come in two types:

1. *cis*-regulatory RNAs. Regulatory RNAs are a part of the mRNA that can regulate how or when the mRNA is used.

2. non-coding RNA genes (ncRNA genes). Most genes encode proteins, but not all. Like protein-coding genes, ncRNA genes are also copied into RNA molecules. However, the RNA molecule is not a messenger used to create a protein; rather, it has its own function. ncRNAs do many of the types of tasks that proteins do.

1.1.5 RNA secondary structure

The need to account for RNA secondary structure is a property that makes computational work with RNA challenging. RNA sequences often contain insufficient information to search for evolutionarily related sequences, in part because RNAs are typically short and over an alphabet with only four letters. Meanwhile much (often more) information is contained in the structure.

As we will see later in this dissertation, RNA secondary structure can be modeled by context-free grammars. Modeling RNA secondary structure is hard because the usual techniques (e.g., regular grammars) are inadequate, yet with large sequence databases, more sophisticated techniques (e.g., context-free grammars) are expensive.

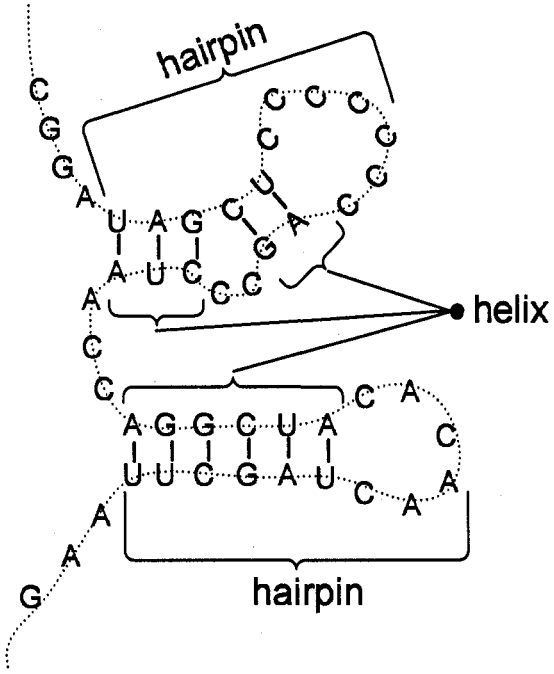


Figure 1.1: Example of RNA molecule secondary structure. The short, thick, solid lines represent hydrogen bonds joining base pairs. The dotted line connects consecutive nucleotides in the linear RNA molecule.
The hypothetical RNA molecule shown has 3 helices, where each *helix* is a run of consecutive base pairs. The upper two helices are separated by two single-stranded C's; this is referred to as a *bulge*. Some helices enclose a *loop*, a single stranded region.
A run of helices (possibly with bulges, as here) and one terminal loop is called a *hairpin*. In this dissertation, I use the most general definition of a hairpin, allowing it to include any internal loops or bulges. (Internal loops are not shown in the example, but are essentially facing bulges on both sides of a helix.) Many authors restrict hairpins to only one helix with no internal loops/bulges, but some authors allow many (e.g., pre-microRNAs are called hairpins).
Some RNAs, like this one, have multiple hairpins; the two hairpins have independent helices.
The helix shown here has no mismatches (although one G-U pairing), but these exist. The upper two helices are separated by a bulge (the nucleotides CC). In some contexts, it is considered one helix with a bulge in it, instead of two separate helices.

When RNA is created from a gene, it is single stranded. However, nucleotides within the RNA molecule can form base pairs with other nucleotides within the same molecule, causing the RNA to fold upon itself. Within an RNA molecule, its A and U nucleotides can form stable Watson-Crick base pairs, as can C and G. "Non-canonical" base pairs are also possible, mainly G and U.

For the purposes of this dissertation, the set of these base pairs is called the *secondary structure.* Figure 1.1 shows an example secondary structure, as well as giving examples of some patterns created by base-pairing: *helix*, *loop* and *hairpin.* Note that *single-stranded* regions involve nucleotides that do not base pair.

A hairpin is a structure containing at least one helix and one terminal loop. In this dissertation, I define hairpins to allow bulges and internal loops, as described in the figure. More formally, suppose we sort the base pairs in a structure by pairing distance, starting at base pairs farthest apart. In a hairpin, each base pair is contained within the previous base pair. (E.g., pairing of nucleotide positions 10 and 20 is contained within a pairing of 5 and 25.)

1.1.6 Analyzing DNA and RNA: Conservation and Homology

Homology. Two biological sequences are *homologous* if they are similar because they derived from a common ancestor. (A way of assessing similarity is explained shortly. The threshold for being "similar enough" depends on the application.) Homologous genes typically have the same or a related function, i.e., if one gene does X, its homologs are likely to do that too.

In general, when two sequences are significantly similar, it is customary to infer that they are homologous. The assumption is that it is unlikely that the similarity arose by chance (i.e., random, unselected mutations), but rather arose because they share a common ancestor. Edit distance (like in the UNIX 'diff' program) is often used to assess similarity.

Conservation. In the process of evolution, DNA sequences will mutate. However, DNA sequences that lead to unfit (or less fit) organisms will generally not survive. If we look at DNA sequences of two related species (say human and mouse), some regions will be highly

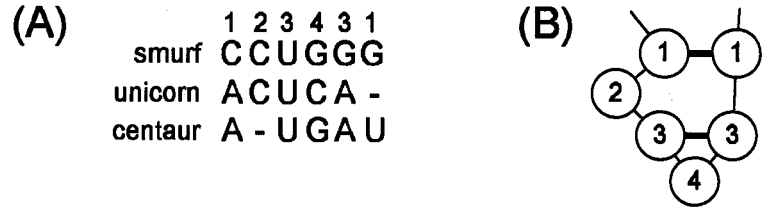


Figure 1.2: RNA multiple alignment. (A) A multiple alignment of three hypothetical RNA sequences, from three hypothetical species. Dashes (-) indicate spacers, to allow related nucleotides to line up. (B) Indication of the conserved RNA secondary structure. Thick lines are conserved base-pairs. Numbers refer to alignment positions; positions 1 and 3 are base paired, so appear twice.
Note that this hypothetical example is unrealistically small. For example, functional RNAs are usually at least 20 nucleotides long, and contain at least two nucleotides in loops (position 4 in this diagram).

similar, while others will be less similar. The most likely explanation is that the highly similar regions are so similar because changes compromise their correct functioning. (Other explanations are possible; this is just the most common.) For example, some changes to a gene may abolish its ability to code for a protein; if the gene is critical, we would not expect to see this change in extant species.

Highly similar sequences between mouse and human are said to be *conserved.* Conservation may also happen at the level of RNA secondary structure. If it is important for an RNA to have a particular secondary structure, then we would expect that structure to be conserved. Note that this can allow changes in sequences. For example, the sequence AAAACCCCUUUU may form a simple hairpin with the AAAA and UUUU base pairing to form a helix, while CCCC is a single-stranded loop. The sequences UUUUCCCCAAAA or GGGGAAAACCCC or even GCGCAAAAGCGC may have the same secondary structure, even though their sequences are different.

A *multiple sequence alignment (MSA)* lines up conserved nucleotides in multiple RNA sequences. For RNAs, MSAs usually indicate the conserved secondary structure. See Figure 1.2 for an example.

1.1.7 BLAST and the Smith-Waterman algorithm

The Smith-Waterman algorithm [115] uses a variant of edit distance, usually with biologically derived scores for the edit operations. To compare the similarity of sequences of length m and n, it uses time $O(mn)$. Since biological sequences are often very large and numerous, this is often too slow.

The *BLAST* [1] algorithm uses heuristics to improve the performance. Primarily, it first looks for a substring of 11 nucleotides that is exactly contained in both the query and the database sequence. (The length parameter does not have to be 11, but can be adjusted.) When it finds this, it tries to extend the match using a variant of the Smith-Waterman algorithm. The speed of BLAST comes at a sometimes unfortunate cost in sensitivity, because exact matches (e.g., of 11 nucleotides) are a strict requirement in many contexts.

1.1.8 Summary

DNAs are long strings over {A,C,G,T}. Parts of DNAs are called genes. Genes are copied into an mRNA, which is then usually made into a protein. ncRNAs are: (1) mRNAs that do more than just carry protein-coding instructions or, (2) RNAs that are not mRNAs at all, but have non-coding functions.

Computational analysis of RNAs is complicated by RNA secondary structure, which is the set of base pairs formed within the molecule. RNA sequences typically contain much information in their structure, so sequence-only methods are at a significant disadvantage. Modeling secondary structure is hard because faster techniques like those based on regular grammars are not applicable.

Two RNAs are homologous if they share a common ancestor. Homologous RNAs tend to have the same function and similar properties. The Smith-Waterman algorithm detects homology using a biological version of edit distance. BLAST is a faster version that matches short, exact subsequences before trying to use a slower Smith-Waterman-like algorithm. Both tools may fail on RNAs because they do not account for RNA secondary structure.

Different parts of an RNA may be more or less tightly conserved, in both sequence and secondary structure. More conserved aspects are generally more biologically crucial to

correct RNA function.

1.2 Stochastic Context-Free Grammars (SCFGs) for non-computer scientists

Context-free grammars (and other formal grammars) were designed in the context of linguistics to try to mathematically model natural languages (e.g., English) [20]. Context-free grammars are now routinely used to define computer programming languages. Their probabilistic version, *Stochastic Context-Free Grammar* (*SCFG*), has proven useful primarily in Natural Language Processing (having computers understand audio speech and text in natural languages) and in modeling RNA sequence and secondary structure.

Context-Free Grammars (CFGs). A context-free grammar (CFG) is a mathematical formalism that can decompose RNA molecules into simple components, analogously to how English grammar can decompose English sentences into words, phrases and parts of speech. In the following, I ignore the full generality of CFGs (and synonyms of the various terms), to focus only on RNA. For RNAs, a CFG consists of:

- The four nucleotides {A,C,G,U}.
- *States*, which represent high-order constructs such as a hairpin or a whole RNA molecule. (In the context of a language such as English, the states would represent subject, predicate, phrases, the whole sentence, etc.)
- *Rules.* The rules of a grammar control how nucleotides and states are put together into an RNA molecule.
- *Start state.* The start state is a special state that represents the whole RNA molecule. By applying rules in various orders, beginning at the start state, a large set of RNA molecules can be created.

The process of transforming the start state into an RNA molecule using the rules is called a *parse*.

Context-Free Grammars can model RNA secondary structure. The types of rules allowed in context-free grammars are powerful enough to express the secondary structure of RNA.[1] The following unrealistically simple grammar represents a helix that is composed only of the Watson-Crick pairs A-U and C-G, with a loop containing only the nucleotide A. Following convention, capital letters are used for states, and lower case letters are used for the nucleotides. The start symbol is S, and states are S (start state), H (helix) and L (loop). The rules are as follows, where "$S \rightarrow H$" means that state "S" can be replaced by "H"; by choosing the right replacements (i.e., rules), we can build a given RNA step-by-step.

$S \rightarrow H$ start at helix

$H \rightarrow L$ helix can stop, going to the loop

$H \rightarrow aHu$ base pair in the helix

$H \rightarrow uHa$ other type of base pair

$H \rightarrow cHg$ other type of base pair

$H \rightarrow gHc$ other type of base pair

$L \rightarrow aL$ only A's can occur in the loop

$L \rightarrow a$ loop can stop

Here is an example parse: $S \rightarrow H \rightarrow aHu \rightarrow aaHuu \rightarrow aacHguu \rightarrow aacLguu \rightarrow aacaLguu \rightarrow aacaaLguu \rightarrow aacaaaguu$. This creates the RNA molecule AACAAAGUU, with the outer AAC and GUU base paired.

CFGs representing more sophisticated secondary structures are also possible.

Stochastic Context-Free Grammars for RNA secondary structure. We have just considered a toy context-free grammar, for a very restricted set of RNAs. But perhaps some of the base-pairs are more common than others, e.g., C-G base pairs are somewhat more stable than A-U base pairs. In other words, any base pairs are possible in the helix

[1] There are cases called *pseudoknots* that cannot be represented by CFGs. Although it would be advantageous to model pseudoknots, this limitations of CFGs does not seem to be severe [33], and more powerful algorithms that include pseudoknots would likely be much slower.

of our stem loop, but some base pairs are more likely than others. We can express this by attaching probabilities to the grammar's rules. The result is a *Stochastic Context-Free Grammar* (*SCFG*) [27].

For example, the following grammar says that helices have 75% C-G pairs on average. ("75%" is arbitrary, not based on any biology.) The grammar also says that helices have a 20% probability of stopping at each nucleotide, and loops have a 25% probability of ending after each A (also arbitrary probabilities).

$$S \rightarrow H \quad \text{p=1}$$

$$H \rightarrow L \quad \text{p=0.2}$$

$$H \rightarrow aHu \quad \text{p=0.1}$$

$$H \rightarrow uHa \quad \text{p=0.1}$$

$$H \rightarrow cHg \quad \text{p=0.3}$$

$$H \rightarrow gHc \quad \text{p=0.3}$$

$$L \rightarrow aL \quad \text{p=0.75}$$

$$L \rightarrow a \quad \text{p=0.25}$$

This grammar could produce the string *acagu* like this: $S \rightarrow H \rightarrow aHu \rightarrow acHgu \rightarrow acLgu \rightarrow acagu$. To calculate a parse's probability, we multiply the probabilities of the rules used in the parse. Thus, the probability of the preceding parse is $1 \times 0.1 \times 0.3 \times 0.2 \times 0.25 = 0.0015$.

The way that SCFGs define probabilities limits the probability distributions that can be represented easily, especially distributions over the lengths of loops. Unfortunately, more sophisticated distributions would lead to slower algorithms. Many successful tools in computational biology share this limitation, e.g., BLAST, Smith-Waterman, profile HMMs.

SCFGs define many algorithms that are useful for RNA analysis. The relevant algorithms will be described in the technical chapters of this dissertation. However, the most commonly used ones are in homology search (what is the likelihood that a given sequence is a homolog of a known RNA or RNA family?) and secondary structure prediction (given an RNA

sequence, what is its most likely secondary structure?).

Chapter 2

INTRODUCTION

Non-coding RNAs (ncRNAs) are functional RNA molecules that do not code for proteins. ncRNAs have been known for decades, e.g., tRNAs and spliceosomal RNAs, but were generally regarded as isolated examples. However, the last several years has seen a dramatic explosion of discoveries of new ncRNAs and novel functions [119, 55, 30, 56, 72], showing that RNAs do more than just carry protein-coding instructions.

For example, two of the most exciting classes of unexpected ncRNAs are microRNAs and riboswitches, although a myriad of other novel ncRNAs have also been found. microRNAs, found in most plants and animals, regulate the activity of other genes, shutting them down at certain times [60, 45, 126, 80, 52]. microRNAs seem to play a role in development of multicellular organisms. Current estimates have at least 800 microRNAs in humans alone [10], making them almost 3% of the predicted 30,000 human genes.

Riboswitches are parts of mRNAs that bind a specific small molecule [135, 125, 134, 137, 86], e.g., vitamin B_{12}, and regulate the mRNA's own activity based on the presence/absence of the target molecule. For example, if an mRNA encodes proteins used to synthesize vitamin B_{12}, it is wasteful to make these proteins if vitamin B_{12} is already available. A vitamin B_{12} riboswitch disables creation of the protein when it detects vitamin B_{12}. The first riboswitch was found as recently as 2002 [134], perhaps because researchers expected that only proteins would have such a role. Riboswitches are widespread among bacteria. A riboswitch that binds thiamin pyrophosphate has also been found in certain plants, fungi and archaea [120].

ncRNA discoveries show no obvious signs of slowing down, and computers can assist this work, by automating key tasks that may be undesirable or even infeasible for a biologist to do. This dissertation addresses a fundamental task for RNA research: given a family of evolutionarily related RNAs, scan genomes for additional members of the family. Performing

this tasks leverages work that went into discovering an RNA family by automatically finding more examples.

This dissertation focuses on one technique, based on probabilistic context-free grammars, called a *covariance model* (CM) [33, 27, 110]. Briefly, CMs have been shown to yield excellent accuracy in these searches, but are infeasibly slow. The main contribution of this dissertation is to speed CM searches by hundreds of times. In most cases, this speedup can be realized with provable guarantees that no accuracy is lost in the faster algorithms, although in some contexts heuristic solutions that lack this guarantee (and are also developed in this dissertation) are useful. Thus, we now have the ability to perform highly accurate searches for RNA family members using arguably the best-available method, in a practical amount of time.

The next section summarizes the way in which solutions to this task contribute to biological research. Section 2.2 shows that RNA homology search is very challenging due to the need to model RNA secondary structure. Section 2.3 summarizes previous solutions to this task. The next section discusses the contributions of this dissertation in speeding CM searches, and is followed by a description of that and related work. The chapter ends with an outline of the dissertation's remaining chapters.

2.1 The impact of RNA homology search

To learn more about an ncRNA family, it is very useful to find additional family members within genome sequence databases. These new family members benefit biological research in several ways:

- **Reduce effort to find ncRNAs in new—possibly highly diverged—organisms.** By automatically finding known ncRNAs in newly sequenced organisms, homology search allows biologists to work on other tasks. The knowledge of the location of ncRNAs and their sequences can also be used in other studies.

  This annotation helps to expand the known occurrence of the ncRNA to new groups of organisms. As an extreme example, homology searches performed by another group

discovered a bacterial riboswitch in the other two domains of life: eukaryotes and archaea [120].

- **Improve knowledge of structure of the ncRNA family.** Having access to many variants of the ncRNA gives a more accurate idea of which regions vary and which are more conserved. This can help to understand its secondary structure or other information.

- **Suggest new biological knowledge.** In some cases, detection of homologs can spur new biological discoveries. For example, this dissertation work includes the discovery, by RNA homology search, that glycine-binding riboswitches typically occur in pairs, adjacent to one another. This immediately raises the question of why two adjacent riboswitches are needed—what biological role does this serve? In a collaboration, biochemists at Yale determined that the riboswitches use cooperative binding, a sophisticated mechanism previously known only in proteins [86]. Cooperative binding allows the tandem riboswitches to bind glycine molecules cooperatively, in a manner leading to a much sharper response curve. This is the first discovery that a naturally occurring RNA can use this intricate biochemical strategy.

The analogous protein family search problem has received significant attention, with several family annotation databases, e.g., [54, 95, 7, 139, 51]. Most protein family homology search is based on profile HMMs [29], which seems to yield excellent performance in most cases, but many improved algorithms for proteins have been proposed. However, solutions for RNA have been inaccurate, slow, complicated to use, or all three.

2.2 RNA secondary structure makes RNA homology search hard

Homology search is well researched in the context of DNA and protein sequences, but solutions for ncRNAs remain unsatisfactory. The key challenge for RNA homology search is the critical significance of RNA secondary structure.

In this dissertation, secondary structure refers to the set of base pairs formed among nucleotides in one RNA molecule. This secondary structure significantly affects the 3-D

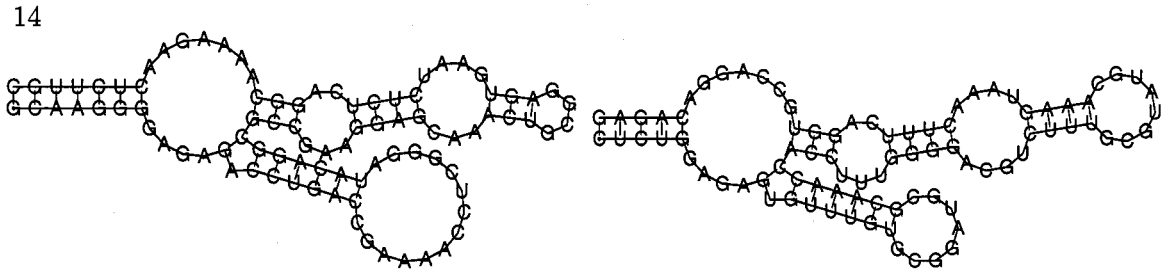


Figure 2.1: ncRNA technology requires modeling secondary structure. Two glycine riboswitches are shown. They appear in tandem in the 5' UTR of yqhI in *B. subtilis*. Using the published alignment, only 29% of columns have identical residues in the two riboswitches. A BLASTN query using one riboswitch fails to achieve a significant E-value even on a database consisting solely of the other sequence. Yet their resemblance in secondary structure is obvious, and both are functional. (Based on [86].)

shape of the molecule, and therefore its biochemical activity.

Sequence-based search algorithms for protein-coding genes are generally inadequate for RNAs. Not as much information is in RNA nucleotides as in proteins' amino acids (mainly since there are only 4 nucleotides, versus 20 amino acids), and RNAs are often short; meanwhile there is much information in RNA secondary structure[1]. Thus, accurate RNA homology searches require attention to secondary structure; for example, see Figure 2.1. Unfortunately, incorporation of secondary structure makes algorithms more complex. This complexity presents a serious challenge, as hundreds of RNA families are known and more are constantly discovered while genome sequence databases expand rapidly.

2.3 Previous work

This section summarizes previous work on finding homologs of a known RNA family. Work can be classified in two orthogonal ways.

- Algorithmic techniques used. For RNA, the techniques used can be classified as either pattern matching or statistical profiles.

- Applicability: is the technique family-specific or generic (applicable to any RNA family)?

[1]Note that protein secondary structure is different from RNA secondary structure; protein methods using "secondary structure" are not applicable to RNA.

After discussing various techniques in these categories, I focus on covariance models (CMs) as a technique that displays excellent accuracy, but is too slow. In the context of CMs, I discuss three CM-based applications: the Rfam Database [48], a large database of ncRNAs that uses CMs; tRNAscan-SE, a family-specific tool that uses CMs; and RSEARCH, which finds homologs of a single RNA.

2.3.1 Family-specific work

Much work has gone into developing tools for searching for a family or a narrow class of RNA. Examples are programs to search for tRNAs [82, 38, 96], microRNAs [81], SRPs [100], tmRNAs [75, 76], SECIS elements [67] and small nucleolar RNAs [83, 35].

This work has proven very useful. For example, the tRNAscan-SE program is used in virtually every genome project to annotate tRNAs. Some specialized search tools for specific families use specialized types of information that are not currently exploited in a generic tool (although the generic tools could be augmented with this information). For example, microRNA finders benefit from requiring conservation in other organisms [81], although there is some reason to believe that many microRNAs (and other RNAs) may not be highly conserved [10]. snoRNAs have short regions that are complementary to rRNAs (sometimes other ncRNAs), and some snoRNA finders [83] use this extra information.

However, hundreds of ncRNA families are now known, and it is undesirable to design family-specific tools for each one. Moreover, research on a new ncRNA often involves iterative refinement of the ncRNA family based on experimental data or on the results of previous homology searches, and each step in this process requires searches for a new ncRNA family; for a novel ncRNA that is inevitably characterized poorly, we need a generic method that can be used for homology searches. Generic methods can also speed development of family-specific tools.

2.3.2 Generic pattern matching work

Techniques for finding ncRNA family members include searching for patterns that can include base pairing [84, 26, 49]. These patterns resemble regular expressions with special

rules for base pairing. (Some pattern languages resemble restricted scripting languages.) These patterns are typically made manually, based on multiple alignments, 3-D structure information and experimental information.

Pattern matching solutions suffer from two problems. First, it can be quite time-consuming to create an appropriate pattern. In practice, creating a pattern requires first inferring a multiple sequence alignment of the known RNAs, or performing biochemical assays, in order to understand what features should be required in the pattern. Then, the pattern must be created manually.

The second problem is that patterns are generally Boolean—they either match a sequence or do not—which can make good sensitivity and specificity difficult. Often, some homologs will violate apparent rules, but otherwise conform to the canonical pattern. If the pattern is loosened to catch these exceptions, the specificity is reduced and true positives are clouded by a large set of false positives. Moreover, slight biases (like a modest overabundance of A or G in one position) cannot be effectively exploited; even if a scoring scheme can be associated with the pattern, it is often too difficult to manually score these slight biases. An algorithm has been developed to optimize patterns made by hand [39], which seems to improve the patterns, but does not address scoring.

One approach to improving RNA pattern-matching algorithms has been to use RNA free energy calculations [67, 124]. It is thought that true RNAs will tend to have lower free energies in their predicted secondary structure (i.e., their structures are more stable). This strategy does seem to reduce false positive rates significantly, but false positive rates are still higher than with statistical profile techniques, and the pattern-based methods still fail to discover ncRNA homologs with anomalous sequence or structure features [124].

2.3.3 Generic statistical profiles

Statistical profile techniques extend weight matrices or hidden Markov models (HMMs) to model RNA secondary structure. Their statistical nature addresses both disadvantages of pattern matching techniques.

First, contemporary statistical profile techniques require as input a multiple sequence

alignment (MSA) annotated with consensus RNA secondary structure. Given such input, they can be created automatically. Creating an alignment is a non-trivial task, particularly since computational solutions tend to be significantly less effective than an alignment manually created by an expert. However, as there is no need to additionally create a pattern, less manual work is required. Moreover, the RNA alignment problem is currently receiving much attention (e.g., see [41, 42]), so it is reasonable to expect substantial improvements in automated alignments.

The second advantage of statistic profiles is that they provide an effective scoring scheme with a sound theoretical basis in statistics. This scoring scheme allows for unexpected violations of the canonical pattern of an RNA. Moreover, these schemes can gain information from even subtle biases in the RNA—and do so automatically.

Two statistical profile techniques exist currently: Covariance models (CMs) [33, 27, 110] and ERPIN [43].

Covariance models. A covariance model is based on probabilistic context-free grammars, where context-free grammar rules are used to model base pairs as well as unpaired nucleotides, based on the input MSA. (How CMs do this is explained in chapter 3.) Grammar probabilities are derived based on frequencies of nucleotides at each position in the MSA. CFG rules also allow for *indels* ("INsertions or DELetions"), which are extra or missing nucleotides relative to the expected conserved nucleotide pattern defined by the MSA. (They are represented as dashes in Figure 1.2.)

Covariance models have been extended with a *local* feature [31, 32, 61]. This feature allows large insertions or deletions at a fixed cost, where otherwise such large indels would be heavily penalized. (It is similar to local alignments, e.g., in the Smith-Waterman algorithm [115].) Local CMs have two advantages: (1) they can accommodate anomalous ncRNA structures, for example, tRNAs that are missing one hairpin of the classic tRNA cloverleaf structure, and (2) they allow the CM to match relatively conserved regions while ignoring highly diverged parts of the ncRNA homolog; highly diverged regions can confuse models that are configured for a fixed evolutionary distance.

CMs are highly accurate. For example, applied to tRNAs, they have a sensitivity of

99.8% with less than 2 false positive predictions per gigabase (estimated on random sequences) [82]. Unfortunately, CMs are extremely slow. For example, a complete scan for the Rfam Database (a database of ncRNAs that uses CMs to scan an 8-Gigabase genome database) would take 10,000 CPU years. Even a single family takes at least one CPU year to scan the 8-Gigabase database used by Rfam. Moreover, new families continue to be found and sequence data expands rapidly.

ERPIN. ERPIN disallows indels in helices (contiguous base-paired positions) in order to facilitate a faster algorithm. Helices are represented by weight matrices over an alphabet of size $4 \times 4 = 16$. ERPIN must consider positions of the left and right halves of each helix, but ignoring indels makes checking each position quicker. ERPIN is able to model some structures containing pseudoknots, a type of secondary structure that cannot be modeled by CMs.

To reduce complexity for ncRNAs with many hairpins, ERPIN stops checking a particular nucleotide position if the first (or later) hairpin's score is below a hairpin-specific threshold. As a result, tuning ERPIN to perform well remains an expert task [73], in addition to the challenge of creating the MSA.

To illustrate differences, CMs and ERPIN become equivalent models if we (1) disallow pseudoknots, (2) avoid the use of "local" semantics, (3) disallow indels in helices, (4) allow ERPIN to consider all hairpins, even if early hairpins score poorly.

Why does work remain on CMs? ERPIN has two advantages over CMs. First, it can represent pseudoknots. However, while a fraction of families do have conserved pseudoknot helices of meaningful size, for most ncRNA families, studies suggest that pseudoknots typically contain little information [33]. Second, ERPIN is significantly faster than CMs—a huge practical advantage.

CMs have three advantages over ERPIN. First, CMs have local alignments, which can improve their ability to find homologs [61]. It is possible that a similar feature could be added to ERPIN, however.

Second, there is no need for per-hairpin heuristics. Both CMs and ERPIN require an

input MSA, which is non-trivial to produce. However, ERPIN has additional input to tune searches, which currently requires expert knowledge of its algorithm. This requirement is practical for a modest number of families curated by experts. However, since CMs require no insight into their detailed algorithms, development of CMs can feasibly be performed by general bioinformaticians or biologists who are performing their own ncRNA research.

It is possible, of course, that ERPIN could be improved, but per-hairpin pruning seems difficult to avoid for medium to large RNAs. ERPIN's algorithm for single-stranded regions is of similar complexity to CM scans, since ERPIN must match the regions beginning at each nucleotide position; this adds an extra factor of complexity. (HMM algorithms, which are applicable here, match prefixes not substrings.) Moreover, combining hairpins is complex, and likely exponential in the number of hairpins if pseudoknots are to be supported. It will be challenging to overcome these issues while maintaining ERPIN's accuracy and without burdening the user.

Third, CMs are able to model indels within helices, unlike ERPIN. Not only is ERPIN unable to model such indels, but they would in most cases be penalized (although if homology is reasonably strong elsewhere, the homolog can still be recognized). ERPIN appears very accurate in practice for tRNAs, the SECIS element and iron response elements [43], but the lack of indels is likely to become more significant when studying ncRNAs that are poorly characterized and perhaps newly discovered. Such ncRNAs are more likely to have unexpected indels, a factor that is easy for CMs to account for. A related issue is that many helices vary in length. Although CMs do not explicitly handle variable-length helices, they are routinely modeled as consecutive positions that are either base paired or missing. Such a feature is difficult to model in ERPIN, particularly given that the variable-length helices are often approximate, and contain indels.

It is clear that ERPIN makes reasonable assumptions, given the practical importance of speed, and its accuracy is very good on well-characterized ncRNAs [43]. At the same time, the technology for RNA homology search is unsatisfactory. If CMs could be made to run in a practical amount of time, without overly sacrificing accuracy or placing additional burdens on the user, this would represent a significant alternative tool for ncRNA searches.

2.4 Applications based on CMs

The Rfam Database [48, 47] is a database of curated multiple alignments of hundreds of ncRNA families. As of March 2005 (Release 7.0), there are 503 families. The curated alignments are used to automatically create a CM; thus, the March 2005 Rfam Database has 503 CMs. Each of these CMs is used to annotate a roughly 8-Gigabase sequence database called RFAMSEQ [48]. RFAMSEQ is a subset of the EMBL nucleotide database [118] that discards lower-quality sequences, such as high-throughput sequencing efforts or incomplete whole-genome shotgun reads.

To run all 503 CMs on RFAMSEQ would take an estimated 10,000 CPU years. Therefore, the Rfam Database uses a heuristic based on the BLAST program to speed searches. (The actual use of BLAST will be explained later in this chapter.) I will show that this BLAST-based heuristic suffers from poor sensitivity in many cases.

tRNAscan-SE [82] uses CMs, and is applied to annotate tRNAs in most genome projects. To improve on the CM's speed, tRNAscan-SE uses two programs previously created specifically for tRNA searches [38, 96]; if either of these programs reports a possible tRNA, the CM is run.

tRNAscan-SE is tRNA specific, but its performance in terms of sensitivity and specificity on tRNAs is excellent [82]. Its speed compares favorably to other filtering methods.

Additionally, tRNAscan-SE has a number of features that are advantageous for tRNA annotation, such as the ability to predict the tRNA's codon. These features are not considered in this dissertation, since the goal of this dissertation is to improve the speed of CMs—something that is independent of these other features.

The RSEARCH program [61] accepts as input a single RNA sequence annotated with a secondary structure, instead of the multiple sequences normally used by CMs. RSEARCH then creates a CM with sophisticated priors trained on curated RNA alignments to search for homologs. Since only one sequence is given, the priors are particularly important. RSEARCH uses local CMs. Highly diverged homologs can be found using RSEARCH, e.g., finding *E. coli* tRNAs using a frog tRNA as a query, or finding SRPs in *Arabidopsis* or archaea, given a human or bacterial SRP as a query. RSEARCH does not use filters, so is

extremely slow; this dissertation presents filters for all CMs, including RSEARCH.

2.5 Contribution of this dissertation

This dissertation improves the speed of CMs using sequence filters, i.e., filters that direct the time-consuming CM to be run on only the most promising sub-sequences. This work makes four contributions:

1. Development of *rigorous filters*, which never compromise sensitivity (described below).

2. Development of a novel type of *heuristic filter*, with better accuracy than other heuristics.

3. Evaluation of heuristic filters on real biological data.

4. Application of filters to biological research, contributing to novel discoveries.

2.5.1 Contribution: rigorous filters

Rigorous filters make a provable guarantee that they will never discard any sub-sequence that the CM would have recognized as a homolog. In other words, the use of a rigorous filter cannot degrade sensitivity. The primary advantage of this technique is that it definitively finds everything that the CM can find, and eliminates the worry that some more sensitive filter may have found another homolog. Rigorous filters described in this dissertation typically improve search speeds by two orders of magnitude. I designed algorithms to create rigorous filters automatically from a CM, so their use imposes little complication to the user.

In addition to their guaranteed perfect sensitivity, rigorous filters have two other advantages. Rigorous filters help to evaluate heuristic filters, because it is possible to know what (if anything) the heuristic filter was missing. Additionally, the rigorous filters developed here may suggest techniques that could be used to speed algorithms other than CM scans, also with guarantees that sensitivity is not compromised.

Rigorous filters are described in chapters 4 and 5, and for local CMs in chapter 6.

2.5.2 *Contribution: the ML-heuristic filter*

A disadvantage of rigorous filters in many contexts is that they can be overly cautious; because they must make guarantees, they may run the CM on too much of the sequence, needlessly slowing searches. I therefore developed a new type of heuristic filter called the *ML-heuristic* ("Maximum Likelihood-heuristic").

The creation of an ML-heuristic filter from a CM is automatic, so any user who has a CM is immediately able to also use an ML-heuristic filter. Although tuning of parameters is possible, empirical results indicate that default settings yield excellent accuracy and practical run times.

The ML-heuristic is described in chapter 7, and extended to local CMs in chapter 8.

2.5.3 *Contribution: evaluation of heuristic filters*

I evaluated various heuristic filters for CMs empirically on real biological data. The results indicate that the ML-heuristic sacrifices only a modest amount of sensitivity, yet is often much faster overall than rigorous filters. Moreover, sensitivity is closely comparable to the popular tRNAscan-SE program, which exploits a large body of research on tRNA detection to design filters specifically for tRNAs. In contrast to tRNAscan-SE, however, the ML-heuristic is generic to any ncRNA family.

As expected, BLAST's sensitivity is often poor. Indeed, for local CMs searching for highly diverged RNAs, BLAST's sensitivity is often zero.

The evaluation of heuristic filters is in chapter 7. Filters for local CMs are evaluated in chapter 8.

2.5.4 *Contribution: application to biological discoveries*

Thanks to these developments, we have highly accurate techniques to find RNA family homologs that requires a practical amount of computer time to run, and a practical amount of human time to prepare its input. This technology has been a part of two important biological developments: the first discovery of a naturally occurring RNA that uses cooperative binding [86] and the realization that 6S RNA is found in virtually all known groups of bac-

teria [6], and not just the γ-proteobacteria group to which it was thought to be restricted. Moreover, the techniques described in this dissertation are available in a freely available software package RAVENNA [129], so further discoveries are expected.

Applications are discussed in chapter 10.

2.6 Summary of sequence filters

In this section, I summarize CM filters, including those from previous work and those introduced in this dissertation. For each one, I give key strength and weakness, motivating rigorous filters and the ML-heuristic. A more complete discussion is in chapter 12

2.6.1 BLAST heuristic

As noted above, the Rfam Database uses BLAST [1] as a heuristic sequence filter for a CM [48]. In this scheme, known ncRNA family members are BLASTed against the RFAMSEQ sequence database (or any other database we wish to scan). Thus, BLAST will find subsequences in RFAMSEQ that approximately match a known ncRNA family member. The CM then scans the approximate matches found by BLAST. Overall scans are accelerated because the CM is run only on a fraction of the original database.

This heuristic assumes that any ncRNA homologs will be similar enough in their sequence features to some known ncRNA family member that BLAST will find them. BLAST may also find many extraneous sequences, but the CM will eliminate these.

Unfortunately, as I show, BLAST often suffers from poor sensitivity—not a surprising outcome, given that BLAST uses no secondary structure information. Because of this, BLAST is not a satisfactory solution in general for CMs filters.

2.6.2 tRNAscan-SE and family-specific filters

tRNAscan-SE was described above, since it is a family-specific RNA homology search tool. tRNAscan-SE inherits the excellent sensitivity and specificity of the CMs it uses. The filters in tRNAscan-SE make it relatively fast, compared to other statistical profile tools.

The main disadvantage is that tRNAscan-SE cannot be used for other families. Moreover, tRNAscan-SE uses two relatively mature family-specific detection programs for its heuristic filter. These two programs represent a substantial amount of tRNA-specific work; indeed, many tRNA detection programs preceded these two, and this kind of effort may be necessary to create the quality of filters used by tRNAscan-SE. Thus, using this strategy for another ncRNA family that lacks work on detection programs may require a substantial investment. Few other RNA families have any family-specific detection programs at all. This strategy is not viable for a general-purpose CM filter.

2.6.3 Rigorous filters

Rigorous filters guarantee perfect sensitivity, which eliminates the need to design better heuristic filters—we know that nothing can be better. This is a key advantage.

The main disadvantage of rigorous filters is that they are often slower than heuristic filters can be, since they must guarantee sensitivity at all costs. As I show, the ML-heuristic's sensitivity is close to 100%, yet a heuristic scan can be 10 times faster than a rigorous scan. (Results refer to Table 7.1 in chapter 7.) So, heuristics can have practical advantages.

Note that any rigorous filter can be used as a heuristic filter, and can be made to be more selective without regard to rigorousness (by adjusting its score threshold independently of the CM's). However, using the filter in this way makes it—by definition—not a rigorous filter. In my evaluation of heuristic filters, I will consider the use of the underlying technology of rigorous filters in a heuristic context.

2.6.4 ML-heuristic

In the ML-heuristic, a profile HMM is created, in which HMM transition and emission probabilities are set to make the HMM maximally similar to the CM.

Overall, the ML-heuristic seems the best available heuristic, since it can easily be applied to any ncRNA family (unlike tRNAscan-SE), can run much faster than rigorous filters and displays better sensitivity than BLAST; the three alternatives each have a significant

disadvantage in general use. However, there is room for developing better filters, as described in future work (chapter 11).

2.7 Outline of this dissertation

This dissertation is organized as follows:

- Chapter 3 explains a simplified version of CMs that will be used to streamline the technical explanations. (However, each chapter describing novel techniques will later indicate how the algorithms work on fully general CMs.)

- Chapter 4 develops rigorous filters based on profile HMMs. Highlights: a context-free-to-regular grammar transformation; use of linear inequalities to enforce rigorousness; a convex optimization problem to design filters that are aggressive yet rigorous (the "infinite-length forward algorithm"); discovery that BLAST was missing many homologs.

- Chapter 5 develops more powerful rigorous filters, augmenting profile HMMs with limited secondary structure information to handle harder families. Highlights: chaining rigorous filters for good overall speed, starting with fast filters and moving to slower but more powerful filters; mixing of profile HMMs and CMs; careful use of extra states to store partial structure information; testing of BLAST and tRNAscan-SE; large-scale, fully rigorous genome scans are practical for all but 2 ncRNA families.

- Chapter 6 extends profile HMM rigorous filters to local CMs. Highlights: an $O(N \log \log M)$ algorithm for emulating certain features of local CMs that would naïvely be $O(NM)$; a new method to optimize rigorous profile HMMs based directly on minimizing the expected Viterbi score.

- Chapter 7 introduces the ML-heuristic. Highlights: the ML-heuristic, a probabilistic transformation of context-free to regular grammars; method to compare heuristic filters' performance; comparisons to BLAST show BLAST discriminates poorly; comparisons to tRNAscan-SE show the ML-heuristic performs comparably.

- Chapter 8 extends the ML-heuristic to local CMs and RSEARCH CMs. Highlights: a method to transform CMs using arbitrary scores into probabilistic models; BLAST's sensitivity is often zero for diverged RSEARCH searches; expanding over-constrained families using RSEARCH, e.g., discovery of T-boxes, previously known only in gram-positive bacteria, in the distant gram-negative bacteria.

- Chapter 9 considers a class of discriminative methods to train heuristic profile HMMs. These techniques appear to provide superior performance for some well-characterized ncRNA families, but are not robust in poorly characterized ones. However, the concepts may provide a foundation for future work. Highlights: method to randomly sample possible homologs from a CM within a restricted range of scores; a discriminative method to train profile HMMs.

- Chapter 10 describes selected applications of the dissertation work to actual biological problems. Highlights: glycine-binding riboswitches use cooperative binding; 6S is found in most bacteria, not just γ-proteobacteria.

- Chapter 11 outlines possible future directions of research.

- Chapter 12 concludes.

Chapter 3

A PEDAGOGICAL VERSION OF COVARIANCE MODELS

Covariance Models (CMs) are statistical models that can detect when positional sequence and secondary structure resembles a given multiple RNA alignment. The following pedagogical model of CMs is used to describe all algorithms in this dissertation.

For simplicity, I ignore RNAs with multiple hairpins (which require CM bifurcation states) and nucleotides inserted into the consensus alignment in the main body of this dissertation. Moreover, I explain CMs somewhat unconventionally in terms of stochastic context-free grammars (SCFGs), and my simplified scheme has a slight limitation relative to true CMs. All algorithms in this dissertation can be extended to true CMs, as they are described in the literature [27, 31]. Furthermore, results are based on RAVENNA, my implementation of my algorithms which uses true CMs (usually taken from the Rfam Database). Each chapter in this dissertation shows how its techniques extend to conventional CMs.

This chapter begins by explaining non-local CMs, ignoring the question of how its grammar rules are created. I then explain the creation of rules, and then describe extensions for local CMs.

Readers unfamiliar with context-free grammars may find section 1.2 or [27, chpt. 9] helpful.

3.1 Covariance Models as context-free grammars

Consider (unrealistically small) RNA molecules with sequence CAG or GAC with the C,G bases paired. A context-free grammar (CFG) for this is $S_1 \rightarrow cS_2g|gS_2c$ and $S_2 \rightarrow a$. (By convention nucleotides in the CFG are lowercase.) S_1 and S_2 are called *states*, and in this case S_1 is the *start state*. The first rule says that S_1 may be replaced by either cS_2g or gS_2c. Thus, we can produce the string CAG by the following steps, beginning with the start state:

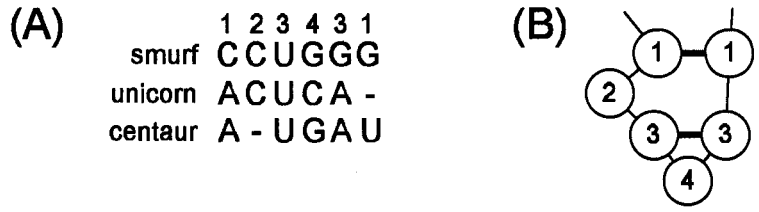


Figure 3.1: RNA multiple alignment, structure and CFG. (A) A multiple alignment of three hypothetical RNA sequences. Dashes (-) indicate missing nucleotides. (B) The structure. Thick lines are conserved base-pairs. Numbers refer to alignment positions; positions 1 and 3 are base paired, so appear twice. Position 1 is missing a base in unicorn.
A CM-style CFG that encodes these sequences and structures is $S_1 \rightarrow aS_2\epsilon|aS_2u|cS_2g$; $S_2 \rightarrow \epsilon S_3\epsilon|cS_3\epsilon$; $S_3 \rightarrow uS_4a|uS_4g$; $S_4 \rightarrow cS_5\epsilon|gS_5\epsilon$; $S_5 \rightarrow \epsilon$. Note that normally CMs use less rigid grammars that allow anomalous nucleotides (with lower probability). A parse of the unicorn sequence is $S_1 \rightarrow aS_2\epsilon \rightarrow acS_3\epsilon\epsilon \rightarrow acuS_4a\epsilon\epsilon \rightarrow acucS_5\epsilon a\epsilon\epsilon \rightarrow acuc\epsilon\epsilon a\epsilon\epsilon = acuca$.

$S_1 \rightarrow cS_2g \rightarrow cag$. The sequence of steps from the start state to an RNA sequence is called a *parse*.

CMs have states $S_1, S_2, \ldots, S_n$ for each of n (possibly base-paired) alignment positions. CFG rules of a restricted form codify sequence and structure characteristics. Methods to construct these rules from an input multiple alignment have been described previously [33, 31, 32], and will be sketched later in this chapter.

All rules must be of the form $S_i \rightarrow x_L S_{i+1} x_R$, where x_L (left nucleotide) and x_R (right) may either be a nucleotide (a,c,g,u) or the empty character ϵ, which produces no nucleotide. If x_L and x_R are both nucleotides, the rule emits paired nucleotides. If $x_L = \epsilon$ or $x_R = \epsilon$ or both, the rule emits an unpaired nucleotide or no nucleotide; such rules can accommodate deletions (i.e., missing consensus positions) and single-stranded regions. In CMs, rules for all possible nucleotides or pairs are used for each state, but rules for unobserved cases have lower probabilities.

Figure 3.1 gives an example RNA multiple alignment and structure, and shows how the rule types above can be combined to create sequences with that structure.

3.2 Genome annotation with CMs

In an SCFG, each rule has a probability. Rules more consistent with an ncRNA family will have higher probabilities than less plausible rules. A parse's probability is the product of the probabilities of the rules used in that parse. For example, if $\Pr(S_1 \rightarrow cS_2g) = 0.25$ and $\Pr(S_2 \rightarrow a) = 1$, then the probability of the parse $S_1 \rightarrow cS_2g \rightarrow cag$ is $0.25 \times 1 = 0.25$. Instead of probabilities, CMs usually employ odds ratios, relative to a simple background model. For computational convenience, the logarithm of the odds ratio is used; the score of a parse is the sum of the logarithmic scores for the rules used in the parse.

For each genome database subsequence, the highest-scoring, or *Viterbi*, parse is computed by dynamic programming [33, 27]. If a subsequence's Viterbi score exceeds a user-supplied threshold specified for a gene family, that subsequence is considered a member of the family.

The use of only one parse (the Viterbi parse) may seem abrupt, but it is an approximation that is commonly used in computational biology and other fields [27]. The Viterbi parse dominates other parses, because probabilities are multiplicative, and thus is usually a reasonable approximation to the true probability. The calculation of the true probability is performed for SCFGs by the Inside algorithm, which is slower and has more numerical issues than the logarithm of the Viterbi.

In the standard CM framework, the user must define a *window length*, which is the maximum length (in nucleotides) that a member of a particular family can be. The window length is a factor in run time for CM scans, so it is important to use a low number—but not so low that homologs are missed. Window lengths in Rfam are typically 100-350.

3.3 Creation of CM rules from an MSA

(The material in this section is used only by the chapters on heuristic profile HMMs, i.e. ML-heuristic. For other chapters, it is not necessary to know how CM rules are created.)

CM rules are created from a multiple sequence alignment (MSA) with annotations indicating which columns are base paired [33, 31, 32]. For base-paired columns, $S_i \rightarrow x_L S_{i+1} x_R$ rules are created. For unpaired columns, $x_R = \epsilon$.

Given the set of rules, it remains to determine what the probabilities of the rules should be; these probabilities should model the information in the input MSA. Each CM rule's probability is set based on how frequently it is used in parsing the MSA sequences. A maximum-likelihood estimate derives a probability estimate by counting the number of times the rule is used and dividing by the counts of all rules with the same CM state in their left-hand side [27]. To avoid zero-probability rules, however, pseudocounts are typically used; all rules have 1 count added before considering the MSA. (More sophisticated priors are starting to become common in CMs.)

In general, CMs allow multiple parses to produce the same sequence, with each parse corresponding to a different secondary structure. For example, consider the following simple CM: $S_1 \rightarrow aS_2|aS_2u$; $S_2 \rightarrow gS_3|gS_3u$; $S_3 \rightarrow \epsilon$. There are two ways to parse AGU: either an A-U pair with free G, or a free A with a G-U pair. However, the input MSA—with its annotated secondary structure, as in Figure 3.1—defines a unique parse for each sequence, because it dictates which nucleotides are paired, and to which consensus position each nucleotide belongs. This makes counting rule uses straightforward.

3.4 Local CMs and RSEARCH

I now extend the simplified CM to handle the local case. I also describe how these local CMs are used in the context of an RSEARCH-like problem [61], i.e., finding homologs of a single query RNA. This definition of CMs is used for the chapters on local CMs, chapter 6 and chapter 8.

Local CMs allow *local begins* and *local ends*. If the start state is made S_0, then local begins use the rule $S_0 \rightarrow S_1|S_2|\cdots|S_N$ to "begin" in the middle of the CM, and find only a part of the RNA. Local ends use rules $S_i \rightarrow S_{N+1}$ for all states S_i, where S_{N+1} matches any RNA sequence, i.e. $S_{N+1} \rightarrow \epsilon|aS_{N+1}|cS_{N+1}|gS_{N+1}|uS_{N+1}$. Local ends gracefully accept missing domains (by replacing state S_i with a small sequence), added domains (by replacing a near-terminal state with a long sequence) and modified domains (by replacing S_i with a long sequence).

Note that this formal definition allows unbounded-length inserts (e.g., 1 trillion nu-

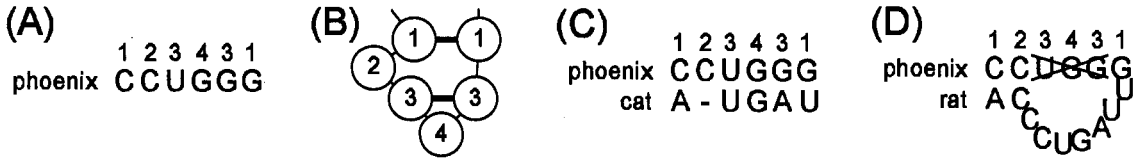


Figure 3.2: RNA structural CFGs: hypothetical RSEARCH example. Suppose a phoenix bird RNA is found that causes the phoenix's mythical immortality. An RSEARCH query is (A) the phoenix RNA sequence and (B) the structure (Thick lines are base-pairs. Numbers refer to nucleotide positions; positions 1 and 3 are paired, so appear twice.) (C) RSEARCH scans find a similar cat sequence, explaining cats' legendary nine lives. The unpaired base in position 2 is missing in cat. (D) RSEARCH also finds a distant rat homolog, in which the inner helix is missing, and is replaced by some other structure; a local end helps to detect this.

Here I show rules for all possible nucleotides. If $\Sigma = \{a, c, g, u\}$, a CFG for the RSEARCH query is $S_0 \rightarrow S_1|S_2|\cdots|S_4$; $S_1 \rightarrow \{\Sigma, \epsilon\}S_2\{\Sigma, \epsilon\}|S_6$; $S_2 \rightarrow \{\Sigma, \epsilon\}S_3|S_6$; $S_3 \rightarrow \{\Sigma, \epsilon\}S_4\{\Sigma, \epsilon\}|S_6$; $S_4 \rightarrow \{\Sigma, \epsilon\}S_5|S_6$; $S_5 \rightarrow \epsilon$; $S_6 \rightarrow \{\Sigma\}S_6|\epsilon$. S_0 handles local begins, S_6 local ends. Rule odds ratios are not shown, but rules more consistent with the query sequence or with RNAs in general will have highest odds ratios. For example, $S_2 \rightarrow cS_3$ is highest for S_2, since it matches the phoenix RNA (i.e., the C in position 2). For S_1, $S_1 \rightarrow cS_2g$ has highest odds ratio since it matches the phoenix RNA; other energetically favorable base pairs (e.g., $S_1 \rightarrow aS_2u$) would have the next-highest odds ratios, since they are most likely in RNAs; and non-canonical base pairs (e.g., $S_1 \rightarrow aS_2a$) would be lowest.

cleotides). However, the window length parameter limits the size in practice.

As above, I will ignore bifurcation states for the main part of this dissertation. However, I note that bifurcations enable structures including multiple hairpins (and therefore multiloops), which make local ends particularly powerful.

Figure 3.2 demonstrates the local concept in the context of RSEARCH.

Chapter 4

RIGOROUS FILTERING USING PROFILE HMMS

This chapter describes a rigorous filter for non-local CMs that is based on profile HMMs. The technique is extended to create more powerful rigorous filters in the next chapter.

Many profile HMM parameter settings guarantee rigorous filtering, but some settings eliminate more of the database than others, thus reducing overall scan time. Under the simplifying assumption that genome sequences are adequately described by 0th-order Markov models, the parameters are optimized to better filter sequences. This requires minimizing an exponential function in variables constrained by linear inequalities, and can yield orders of magnitude improvements in filtering over less sophisticated strategies.

In my experiments, the profile HMMs search on average over 200 times faster than CMs. I tested 139 ncRNA families in Rfam version 5.0 (all families except those using the Rfam local alignment feature; these families are dealt with in chapter 6). For 110 of the 139, the profile HMM filtered the database to less than 10^{-4} of its original size. For these families, the total search time will be dominated by the quicker profile HMM.

For some families in Rfam 5.0, no new members were found over the BLAST heuristic. Since the filtering is rigorous, this means that if homologs were missed, finding them will require going beyond CMs as they are now. For other families, rigorous filters find many homologs missed by the BLAST heuristic that are biologically plausible, and potentially add important information. E.g., a small nucleolar RNA of *Pyrococcus* is found to have putative homologs in a variety of other hyperthermophilic archaea.

See `http://bio.cs.washington.edu/supplements/zasha-RECOMB-2004/` for raw results of rigorous Rfam 5.0 scans. Additional technical information is in section 4.4.

Chapter 3 described a simplified version of CMs that I use for the purpose of exposition. However, in section 4.4.6, I show how the techniques work on fully general CMs.

Table 4.1: Filtering fraction of optimized profile HMMs. The fractions are grouped into qualitatively similar ranges, e.g., fractions less than 10^{-4} imply that the time will be dominated by the HMM scan.

x=**filtering fraction**	**# of families**	
	compact-type HMM	expanded-type HMM
$0 \leq x < 10^{-4}$	105	110
$10^{-4} \leq x < 10^{-2}$	8	17
$10^{-2} \leq x < 10^{-1}$	11	3
$10^{-1} \leq x < 0.25$	2	2
$0.25 \leq x < 0.99$	6	4
$0.99 \leq x \leq 1$	7	3

4.1 Results

4.1.1 Speed

The technique's overall speedup over a CM is the sum of (1) the HMM scan time divided by the CM's, plus (2) the fraction of the sequence that the CM is run on. This yields a number less than 1 (or slightly more if HMM filtering is very poor).

The fraction of a test sequence that the CM must be run on, i.e., that is not filtered, is the *filtering fraction.* A fraction of 0 is perfect; 1 is pointless. I tested the filtering fraction of 139 Rfam 5.0 families against two genomes: *E. coli* K12 and *Staphylococcus aureus* MW2, which has a very low G+C content that many HMMs perform less well on. For each family, 2 HMMs were created, one optimized for a maximum likelihood 0th-order Markov model for *E. coli* and one for *S. aureus.* Conservatively, the *E. coli*-optimized HMM was run on the *S. aureus* genome and vice versa. The filtering fraction used is the maximum (worst) for the two genomes. The results, summarized in Table 4.1, show that most families' HMMs eliminate most of the database.

The profile HMM scan itself runs on average over 200 times faster than the CM scan (270 times for "compact" type HMMs, 214 times for "expanded" type; these types are

explained later). For some families the filtering fraction is near 1, so the profile HMM cannot accelerate searches. Taking only the 127 families (out of 139) with filtering fractions $< 10^{-2}$, annotating these families using profile HMM rigorous filters would be 283 times faster than using raw CMs.

4.1.2 Buried treasures

I scanned the full RFAMSEQ sequence for selected Rfam families (Table 4.2). For some families, no additional homologs were found compared to the Rfam database; given the technique's guarantees, it follows that the BLAST heuristic is sufficient for these families. I biased the selection to families that I expected to challenge BLAST, e.g., short ncRNAs with low sequence identity, so I expect that BLAST will work for most other families, which are presumably easier.

The families on which BLAST found all homologs tend to be phylogenetically narrow and their RNA sequence highly similar. Since CMs do not currently have robust priors, the CMs for these families may be overspecific, which would give BLAST an advantage that does not reflect the biology.

Moreover, I also avoided families for which the HMM's filtering fraction was poor. Since neither BLAST nor HMMs use secondary structure, I expect BLAST to do worse for those families. In fact, the next chapter develops more powerful filters that can handle these difficult families, and demonstrates that BLAST does indeed have more trouble with them.

For many families, new hits (i.e., putative homologs) were found with the rigorous filter that were classified as positives by the CM, as summarized in Table 4.2. Some of these hits may be false positives biologically; even so, false positives may aid in tuning the CMs.

Without experimental data, it is impossible to be certain whether these new hits are biologically correct. However, other evidence suggests that while some hits are indeed biologically false, a significant number of new hits are strongly supported by other annotation evidence. This indicates that true homologs were indeed being missed because of BLAST. Moreover, many hits are in novel species, so the added information may significantly expand our understanding of the ncRNA family.

Table 4.2: Results of rigorous filtering experiments of RFAMSEQ. Each row in this table is an Rfam 5.0 family. The first column is its Rfam accession Id. (The first row is from Rfam version 4.0.) Next is a brief name, followed by the average length in nucleotides. The % identity (sequence conservation) is as reported in Rfam. # known is the number of members in Rfam, i.e., members identified from other sources and homologs found using the BLAST heuristic. # new is the number of additional matches in RFAMSEQ that my technique found. For HMM types used, C=compact type, E=expanded type. "C,E" means that a compact-type HMM filtered RFAMSEQ, then an expanded-type HMM further filtered this. The (conservative) predicted filtering fraction on *E. coli* and *S. aureus* is given for the expanded-type HMM, or if no expanded-type HMM was used, for the compact-type HMM; this is the fraction the CM is predicted to be run on. The notation "1.1e-3" means 1.1×10^{-3}. The next column is the actual fraction on RFAMSEQ. The CPU time taken to scan the 8-gigabase RFAMSEQ on a 2.8 GHz Pentium 4 is then reported, then the time for a pure CM scan of RFAMSEQ, extrapolated from the time for a 10 Kb sequence.

Rfam Id	name	avg len	% id	# known	# new	HMM types used	frac. pred.	frac. actual	HMM +CM time (days)	est. CM time (days)
Rfam 4.0 RF00008	Hammerhead ribozyme	35	68	313	278	C,E	1.1e-3	6.1e-4	1.4	97
RF00008	Hammerhead 3	54	78	251	13	C,E	2.8e-3	2.4e-3	3.2	343
RF00012	U3 snRNA	227	59	129	0	C	0	8.9e-6	10.1	3808
RF00015	U4 snRNA	136	60	283	7	C,E	1.6e-3	1.3e-3	8.3	1258
RF00019	Y RNA	102	69	1107	0	C	0	1.4e-5	4.4	524
RF00020	U5 snRNA	118	59	199	1	C,E	1.0e-2	5.0e-3	8.9	1081
RF00024	Vertebrate telomerase	436	60	51	0	C	0	3.5e-5	17.6	10349
RF00025	Ciliate telomerase	168	57	17	0	C	0	3.8e-7	6.3	1588
RF00026	U6 snRNA	106	80	1462	1	C	0	2.4e-5	4.0	563
RF00027	*let-7* microRNA	80	68	30	0	C,E	1.5e-3	1.5e-3	3.7	500
RF00030	RNase MRP	265	52	39	0	C,E	0	9.5e-6	10.4	6498
RF00032	Histone 3' element	26	78	1004	102	C	1.9e-5	2.0e-5	1.1	29
RF00037	Iron response elem.	29	67	201	121	C,E	7.7e-4	7.7e-4	1.0	52
RF00043	Plasmid copy	73	74	8	0	C	0	8.2e-8	2.5	475
RF00050	RFN element	147	67	107	0	C,E	1.7e-4	5.6e-6	4.5	1666
RF00052	*lin-4* microRNA	69	70	14	0	C	0	1.1e-7	2.5	343
RF00053	mir-7 microRNA	87	67	10	0	C	0	6.5e-8	3.4	363
RF00054	U25 snoRNA	80	66	28	0	C	0	2.3e-7	3.8	410
RF00055	snoRNA Z37	92	68	28	0	C	0	2.5e-7	3.2	371
RF00066	U7 snRNA	61	71	312	1	C	1.9e-5	8.4e-6	2.6	231
RF00075	mir-166 microRNA	122	60	14	0	C	0	1.8e-7	4.4	813
RF00093	U18 snoRNA	75	63	43	0	C	0	6.0e-7	3.2	332
RF00095	*Pyrococcus* snoRNA	56	59	57	123	C	0	2.5e-6	2.8	249
RF00103	mir-1 microRNA	77	69	13	0	C	0	1.1e-7	3.0	373
RF00104	mir-10 microRNA	74	67	34	0	C	0	3.8e-7	2.4	358
RF00151	U58 snoRNA	65	85	12	0	C	0	6.2e-8	2.7	156
RF00162	S box	110	66	128	3	C,E	1.7e-3	1.5e-3	6.7	1042
RF00163	Hammerhead 1	82	62	167	26	C,E	1.7e-3	1.1e-3	2.8	473
RF00164	Coronavirus s2m	43	80	115	0	C	0	5.8e-7	1.4	118
RF00165	Coronavirus 3' UTR pseudoknot	62	70	60	0	C	0	4.0e-7	2.9	212
RF00167	Purine element	99	55	69	31	C,E	8.8e-3	4.2e-3	5.5	604
RF00169	Eubacterial SRP	99	52	162	0	C,E	7.2e-4	5.7e-4	4.4	588
RF00170	Retron msr RNA	71	57	11	48	C	3.0e-3	1.1e-3	2.5	289
RF00173	Hairpin ribozyme	51	83	5	0	C	0	3.1e-8	1.7	174

Hits in new organisms. The hits not found with BLAST included candidate homologs in organisms that already have known homologs, as well as potential homologs in new organisms. A small nucleolar RNA (Rfam ID RF00095) known only in *Pyrococcus* species has new hits in 11 other hyperthermophilic archaea, e.g., *Archaeoglobus fulgidus* and *Methanococcus jannaschii.* The retron msr RNA (RF00170), found in a range of bacteria, has three new hits in eubacteria outside of the proteobacteria group that contains all the training family members. The hammerhead ribozyme (RF00008, RF00163) is found in new viral genomes and in repetitive elements, where it has been found previously [37]. The histone downstream element (RF00032), usually in metazoa, is found in various new metazoa and, intriguingly, in three plant species and four bacteria. The iron response element (RF00037), also usually in metazoa, is found in two fungi, various invertebrates, chimpanzee, and with several putative homologs in *Arabidopsis* and rice.

Evaluation of new hits. I use the following types of information to evaluate the new hits: annotations of the genome sequences (including annotations of genes near candidate mRNA regulatory elements), phylogenetic plausibility and tandem repeats annotations, a confounding factor:

- Hits for the Hammerhead ribozyme (RF00008,RF00163) are not contradicted or supported by annotations. As noted above, it is expected to find these ncRNAs in viruses and in eukaryotic repetitive elements as I did, although a large fraction of RFAMSEQ is composed of such sequences.

- Two U4 snRNA (RF00015) hits are annotated as U4: in green algae and a fungus. Two new human hits and one mouse hit are in annotated U4 snRNA-based repeat sequences.

- 15 histone 3' element (RF00032) hits are 3' to an annotated histone gene: in 5 *Drosophila* species, *Tigriopus californicus* (marine invertebrate), *Chironomus thummi* (fly) and *Platynereis dumerilii* (annelid worm). 18 hits are in repeat families, 4 in histone-related repeats. The three plant hits are annotated as internally transcribed

spacers, and may be unrelated hairpins. 6 hits are in predicted coding regions (so are probably false positives), including two of the bacterial hits. Note: many new hits have the same sequence as hits found in Rfam by BLAST; since the histone element is small, I presume that these hits were found only in some cases because of approximate matches in adjacent sequences.

- The iron response element (RF00037) is a protein binding site in UTRs of genes related to iron metabolism. I find two new hits upstream of ferritin genes, as expected, in *Lymnaea stagnalis* (a snail) and *Branchiostoma belcheri* (a marine invertebrate). 13 hits were in regions annotated as coding, three based on experimental evidence. Three other hits were in introns of annotated genes not obviously related to iron. 12 were in repeat regions.

- The U7 snRNA (RF00066) has one hit not found by BLAST, which is annotated as a U7 repeat region.

- A snoRNA previously with homologs known only in *Pyrococcus* (RF00095) is, as above, found in many other hyperthermophilic archaea. 37 new hits are annotated as snoRNA, probably by computer. An additional 5 are annotated as snoRNA based on study of snoRNA. A further 3, in *Archaeoglobus fulgidus*, *Pyrococcus furiosus* and *Sulfolobus acidocaldarius* are experimentally confirmed snoRNAs. On the other hand, 12 hits are in annotated coding regions (including all bacteriophage hits) and one human hit is in an annotated repeat region.

- The S-box (RF00162) is a riboswitch regulatory element that binds S-adenosylmethionine, and is found upstream of methionine- and cysteine-related genes. All three new hits are upstream of genes on the correct strand. The genes are annotated as an unknown ABC transporter and hypothetical protein, both of which might be related to S-adenosylmethionine, and flavin mononucleotide reductase, which is unexpected.

- The purine element (RF00167) is a riboswitch regulatory element controlling purine biosynthesis and transport genes. Three of the new hits (in bacteria) are upstream of

potential purine-related genes: adenine deaminase, xanthine/uracil transporter (xanthine is a purine and a precursor and degradation product of adenine and guanine) and an uncharacterized ABC transporter (which might transport purines). All 28 hits in eukaryotes are not supported by any evidence; moreover, where the confirmed family members have a G+C content of 30-45%, the eukaryotic hits' G+C content is 1-20%—a fact that calls the predictions into question (J.E. Barrick, personal communication).

- The retron MSR RNA (RF00170) is typically found in eubacteria upstream of the reverse transcriptase used for mobility of the retron. However, while there are 3 hits in eubacteria, none are near annotated reverse transcriptases.

To see if BLAST could easily be adjusted to find all new hits, I tested the Rfam family RF00095, which has the most new hits. Of the 110 biologically plausible new hits, i.e., within hyperthermophilic archaea, BLAST is able to find 56 of them if its E-value threshold is raised to 10000, but then its filtering fraction is 0.014, which makes it slower than the profile HMM rigorous filter.

Thus, the rigorous filter finds putative homologs missed by BLAST's heuristic filter, both in organisms with known homologs, and plausibly in new organisms. It thus potentially expands our understanding of these families.

4.2 Construction of the profile HMM from a simplified CM

Given a CM, a profile HMM is created whose Viterbi score for any sequence is always an upper bound on that of the CM. Although profile HMMs are less powerful than CMs, their Viterbi algorithm is much faster, which makes them an attractive filter. I describe HMMs in terms of stochastic regular grammars. First, I describe how I will use the profile HMM as a filter, then I explain the form of regular grammars allowed, then show how to convert a CM's SCFG into such a grammar. To guarantee rigorous filtering, the HMM rules' logarithmic scores are constrained so that the HMM's score for a database subsequence is an upper bound on the CM's score. After describing these constraints, I show how to optimize scores

to filter efficiently, subject to the constraints.

My algorithm can create two related profile HMM structures based on a given CM: the "compact"-type HMM and the "expanded"-type HMM. The expanded-type HMM adds extra states that improve filtering, at the cost of scanning the genome database about 30% slower. The technical distinction between these types of profile HMM depends on the use of fully general CMs, which is discussed in section 4.4.6.

Filtering with the profile HMM

The HMM calculates a CM score upper bound for sequences ending at each nucleotide position in the database sequence. If one of these upper bounds exceeds the threshold, the CM algorithm is applied to a window ending at that nucleotide position. If the HMM-generated upper bound is lower than the threshold, then the CM will provably not report a homolog at that location, so the location can safely be filtered out. (More details are in section 4.4.3.)

4.2.1 Creation of a stochastic regular grammar for a profile HMM

Regular grammars are less powerful than SCFGs in that their rules cannot emit paired nucleotides. Specifically, rules must be of the form $\mathrm{S}_i \rightarrow x_L \mathrm{S}_{i+1}$.

Consider a CM with two states $\mathrm{S}_1, \mathrm{S}_2$ with rules $\mathrm{S}_1 \rightarrow x_L \mathrm{S}_2 x_R$ and $\mathrm{S}_2 \rightarrow \epsilon$. If $x_R = \epsilon$ in all cases, this CM can be represented directly by a profile HMM.

The key challenge is when $x_R \neq \epsilon$ in some rules. For example, suppose we have $\mathrm{S}_1 \rightarrow a\mathrm{S}_2 u | c\mathrm{S}_2 g$. A profile HMM cannot represent the fact that the bases are paired, but can reflect the sequence information by breaking the pair encoded by S_1 into two HMM states: $\overline{\mathrm{S}}_1^L$ handles the left nucleotide and $\overline{\mathrm{S}}_1^R$ the right. (HMM states will be written with a bar to differentiate them from CM states.) Here is a regular grammar: $\overline{\mathrm{S}}_1^L \rightarrow a\overline{\mathrm{S}}_2^L | c\overline{\mathrm{S}}_2^L$; $\overline{\mathrm{S}}_2^L \rightarrow \overline{\mathrm{S}}_1^R$; $\overline{\mathrm{S}}_1^R \rightarrow g|u$. This profile HMM grammar encodes the fact that the first nucleotide is A or C, and the second G or U, although it sacrifices the information that only A-U or C-G pairs are permitted; e.g., it allows A-G. This sacrifice is a necessary consequence of using profile HMMs.

Table 4.3: Example of converting a CM to a profile HMM. The CM grammar of Figure 3.1 is converted to a profile HMM grammar, rule by rule. The HMM can be read in sequential order by going down the middle column, then up the right column.

CM state rules	**left HMM state rules**	**right HMM state rules**
$S_1 \rightarrow aS_2\epsilon \mid aS_2u \mid cS_2g$	$\overline{S}_1^L \rightarrow a\overline{S}_2^L \mid c\overline{S}_2^L$	$\overline{S}_1^R \rightarrow \epsilon \mid g \mid u$
$S_2 \rightarrow \epsilon S_3\epsilon \mid cS_3\epsilon$	$\overline{S}_2^L \rightarrow \epsilon\overline{S}_3^L \mid c\overline{S}_3^L$	$\overline{S}_2^R \rightarrow \epsilon\overline{S}_1^R$
$S_3 \rightarrow uS_4a \mid uS_4g$	$\overline{S}_3^L \rightarrow u\overline{S}_4^L$	$\overline{S}_3^R \rightarrow a\overline{S}_2^R \mid g\overline{S}_2^R$
$S_4 \rightarrow cS_5\epsilon \mid gS_5\epsilon$	$\overline{S}_4^L \rightarrow c\overline{S}_5^L \mid g\overline{S}_5^L$	$\overline{S}_4^R \rightarrow \epsilon\overline{S}_3^R$
$S_5 \rightarrow \epsilon$	$\overline{S}_5^L \rightarrow \overline{S}_4^R$	

In general, a CM state S_i is expanded into a *left HMM state* $\overline{S}_i^L$ and a *right HMM state* $\overline{S}_i^R$. All CM rules $S_i \rightarrow x_L S_{i+1} x_R$, regardless of the value of x_L and x_R, are converted into HMM rules $\overline{S}_i^L \rightarrow x_L\overline{S}_{i+1}^L$ and $\overline{S}_i^R \rightarrow x_R\overline{S}_{i-1}^R$. (Note that the subscript on $\overline{S}^R$ is decremented, since the right nucleotides are emitted in reverse order.) If $i = 1$, then we omit $\overline{S}_{i-1}^R$. The procedure may result in duplicate rules being created; duplicates (if any) are removed. Finally, the last left HMM state is connected to the last right HMM state with rule $\overline{S}_N^L \rightarrow \overline{S}_{N-1}^R$. See Table 4.3 for an example.

Constraints on HMM logarithmic scores

In the previous subsection, I showed how to create an HMM grammar corresponding to a CM grammar. As a first step to assigning logarithmic scores to HMM rules, I now define constraints that ensure that the HMM's Viterbi parse score for any database subsequence is an upper bound on the CM's Viterbi score.

I defined HMM rules that correspond to each CM rule. Any CM parse (Viterbi or not) consists of a sequence of rules, which can be mapped to HMM rules, obtaining a corresponding HMM parse[1]. The score of a parse is the sum of the logarithmic scores of its rules. We could enumerate all possible CM parses and require the sum of scores along the

[1]To be unambiguous, a CM parse must be modeled as a tree of rules. However, this ambiguity is not an issue in mapping parses to profile HMM parses.

corresponding HMM parse to be greater or equal to that of the CM, thus guaranteeing the upper bound.

Unfortunately, the number of CM parses is exponential in the number of states. However, we can enumerate all CM rules $S_i \rightarrow x_L S_{i+1} x_R$ for all i. For each rule, we consider the corresponding HMM rules. The sum of their logarithmic scores must be greater or equal to the CM rule's score. Thus, each CM rule leads to one linear inequality in terms of the logarithmic HMM scores.

For example, the grammar in Figure 3.1 has the rule $S_3 \rightarrow uS_4a$ and corresponding HMM rules $\overline{S}_3^L \rightarrow u\overline{S}_4^L$ and $\overline{S}_3^R \rightarrow a\overline{S}_2^R$. Similarly, $S_3 \rightarrow uS_4g$ corresponds to $\overline{S}_3^L \rightarrow u\overline{S}_4^L$ (again) and $\overline{S}_3^R \rightarrow g\overline{S}_2^R$. Let l_1 be the logarithmic score for $\overline{S}_3^L \rightarrow u\overline{S}_4^L$, l_2 for $\overline{S}_3^R \rightarrow a\overline{S}_2^R$ and l_3 for $\overline{S}_3^R \rightarrow g\overline{S}_2^R$, and suppose the score of CM rule $S_3 \rightarrow uS_4a$ is -1 and $S_3 \rightarrow uS_4g$ is -2. Then we obtain one inequality per CM rule: $l_1 + l_2 \geq -1$ and $l_1 + l_3 \geq -2$, where a trivial solution is $l_1 = -1, l_2 = l_3 = 0$. Every CM rule will result in one inequality, and any solution to the HMM scores (like l_1, l_2, l_3) satisfying all inequalities ensures the upper bound.

These inequalities constrain the selection of values of l_1, l_2, l_3, but some solutions to these inequalities may filter better than others. The HMM rule relating to l_1 is responsible for emitting left nucleotides. For the right nucleotide, l_2 emits As and l_3 emits Gs. If As and Gs are equally probable in database sequences, then on average the contribution to the Viterbi score of this base pair is $l_1 + \frac{1}{2}(l_2 + l_3)$, since l_1 is the only alternative for the left nucleotide, and either l_2 or l_3 may apply to the right nucleotide. If $l_1 = -1$, $l_2 = l_3 = 0$, then the average added score is -1. However, with the solution $l_1 = 0$, $l_2 = -1$, $l_3 = -2$ (also satisfying $l_1 + l_2 \geq -1$, $l_1 + l_3 \geq -2$), the average added score is -1.5, which is lower. Thus, while both solutions satisfy the inequalities, the latter will likely lead to lower HMM scores, and therefore require the CM to be run less often. In the next subsection, I present an approach to selecting scores that uses a more sophisticated analysis of what the expected score will be than simply considering each state in isolation.

4.2.2 Optimizing the profile HMM

The previous subsection defined constraints on the HMM scores, and noted that some solutions to the constraints will filter better than others. In this section, I consider a heuristic to improve filtering. Regardless of the heuristic to improve filtering, I still require that the constraints are satisfied, so that rigorous filtering is guaranteed.

(Note that I use a different method to optimize the profile HMM for local CMs. The algorithm described here worked poorly in the local case. Evaluation of the objective function for the local case, described in chapter 6, is considerably slower than this chapter's algorithm, but produces better HMMs. In the local case, the difference in HMMs' filtering fractions is very significant; in the non-local case, the difference is marginal.)

To improve filtering, we need a method to estimate an HMM's likely filtering fraction. For example, one could measure the HMM's average filtering fraction for a database sequence sampled from some probabilistic sequence model; the optimal HMM scores would minimize this expected fraction. This idea suffers two practical drawbacks. First, it is slow, because a test database sequence must be relatively large to be adequately representative. Second, the expected fraction is not differentiable with respect to HMM scores, eliminating many optimization algorithms, e.g., gradient descent. We need a more practical heuristic.

The infinite-length forward algorithm score I develop a more practical heuristic to approximate the filtering fraction in a series of steps. First, I assume that the database sequence is distributed according to a 0th-order Markov model, i.e., with independent probabilities of A,C,G,U. Next, instead of the filtering fraction, I measure the expected Viterbi algorithm score when run on a 0th-order sequence model; reducing the expected score should result in fewer scores being above the threshold, and therefore a better filtering fraction.

When the HMM is used for filtering, it calculates a Viterbi score over all subsequences ending at a given database position, so obtains a score for each database position. We wish to minimize these scores in the expected case. The subsequences incorporated into each Viterbi score are drawn from the 0th-order model, and could in theory be any length, limited only by the length of the database. I now assume that the database is infinitely long.

This assumption is motivated by the fact that it will lead to a more tractable formula later in the formalization, since there is no need to explicitly limit the length of subsequences. The assumption is reasonable because in practice most database sequences will be significantly longer than an ncRNA homolog could plausibly be; sequences whose length approaches infinity will have paths of exceedingly low odds ratio, so are irrelevant. (Indeed, with simplified CMs, which do not allow for inserted nucleotides, sequences beyond a certain length will have odds ratios of 0.)

The final modification to the heuristic is, instead of the Viterbi score, to use the forward algorithm, which will allow for fast evaluation and yield analytic gradients. Where the Viterbi algorithm finds the highest-scoring parse, the forward algorithm will compute the sum of the compound odds ratios of all possible parses. The forward algorithm is theoretically more accurate, since it takes into account sub-optimal parses. But, the Viterbi is often used since it is more practical, and in practice gives similar results. I take the opposite step, and use the forward algorithm to approximate the Viterbi in estimating an HMM's filtering fraction.

Thus, the heuristic is to use the expected HMM forward algorithm result over database subsequences of unbounded length, generated from a 0th-order model. I presume that a lower expected forward algorithm result will correlate with a lower filtering fraction when the HMM is run on real genome sequences using the Viterbi algorithm. The mathematical details are left to section 4.4.4, which also describes an efficient dynamic programming algorithm to compute the score and its partial derivatives. (These partial derivatives are used by optimization algorithms such as gradient decent.)

Empirically, the infinite-length forward algorithm score is superior to the simple objective function sketched earlier ($l_1 + \frac{1}{2}(l_2 + l_3)$), at least on fully general CMs; it is too simplistic to consider each state in isolation in estimating its effect on the total score.

Optimizing with the infinite-length forward algorithm The optimal logarithmic scores for HMM rules (1) satisfy the inequalities and (2) minimize the expected infinite-length forward algorithm score. For practical reasons, we iterate over each CM state, minimizing the forward algorithm score for HMM score variables corresponding to that

CM state, subject to the inequalities, keeping other CM states' variables fixed. We repeat these iterations over CM states until the score cannot be improved after cycling through all states. The optimization problems are solved with CFSQP[77], though other solvers could be applied. Optimizing both compact- and expanded-type HMMs for 139 models in Rfam 5.0 on a 2.8 MHz Pentium 4 took about 4.5 CPU days.

4.3 Discussion

This chapter showed how rigorous filters can be created, and found that they are very effective at speeding scans. This result was somewhat surprising, given that profile HMMs cannot model secondary structure. Current implementations of CMs use naïve priors (a simple plus-1 pseudocount). Many Rfam families have highly similar sequences; this, in combination with the naïve priors, creates overly specific models—in particular models that overstate the significance of sequence information. It is likely that profile HMM rigorous filters will speed fewer families when this issue is addressed. For the same reason, BLAST will likely miss even more homologs.

I have extended the work in this chapter in three ways:

- The next chapter introduces a more powerful class of rigorous filters, which allow rigorous scans for all but 2 of the Rfam families.

- Chapter 6 extends the profile HMM rigorous filters to local CMs.

- Chapter 7 describes the ML-heuristic, a heuristic also based on profile HMMs.

4.4 Additional details

This section contains additional details not crucial to understanding this chapter:

- Proof that the profile HMMs are indeed rigorous.

- Analysis of differences in profile HMM vs. CM scan run times.

- Details on how profile HMMs are used in filtering.

- Details on the infinite-length forward algorithm: a formal definition and dynamic programming solution; proof that the objective function is convex; details on solving the optimization problem.

- Extension to fully general CMs: definition of general CMs; adaptation of algorithm and proof of rigorousness to this case; definition of compact- vs. expanded-type HMMs.

4.4.1 Proof of rigorousness

In this section I sketch a proof that the HMMs defined in this chapter do, in fact, implement rigorous filtering of the simplified pedagogical CMs. I prove that the logarithmic score that the HMM assigns to any RNA sequence is an upper bound on the logarithmic score that the CM assigns. Given this property, rigorous filtering is easily implemented relative to some family-specific CM score threshold.

Proof idea. We wish to prove that the HMM score for a sequence is an upper bound on the CM score. Let the CM's Viterbi parse of some sequence x be π^{CM}. I constructed the profile HMM grammar in a specific manner; because of this construction, we will be able to find an HMM parse π^{HMM} such that:

1. HMM parse π^{HMM} emits the sequence x, just like the CM parse.

2. HMM parse π^{HMM} has score greater than or equal to that of CM parse π^{CM}.

It follows that the HMM score is an upper bound on the CM score over all possible sequences.

Proof. Given a CM parse π^{CM}, we find a parse π^{HMM} with the foregoing properties. I view a parse as a set of grammar rules[2]. The CM parse is then a set of rules of the form $\mathrm{S}_i \rightarrow x_L \mathrm{S}_{i+1} x_R$.

[2]To avoid ambiguities, in general we must model CM parses as trees. However, since we are only concerned with scores, trees are unnecessary for this proof.

For each rule, we use the corresponding HMM rules $\overline{\mathrm{S}}_i^L \rightarrow x_L \overline{\mathrm{S}}_{i+1}^L$ and $\overline{\mathrm{S}}_i^R \rightarrow x_R \overline{\mathrm{S}}_{i-1}^R$ to construct the profile HMM parse π^{HMM}. All of these profile HMM rules will exist in the profile HMM grammar because they would be generated from the CM rules in constructing the profile HMM. Moreover, by inspection, these profile HMM rules can generate precisely the same RNA sequence as the CM rules they were generated from.

It remains to prove that the score of π^{HMM} is at least as high as π^{CM}. I imposed inequality constraints as part of the optimization of profile HMMs based on the infinite-length forward algorithm. Because of these constraints, if $\mathcal{S}(.)$ is the score of a rule, we know that $\mathcal{S}(\overline{\mathrm{S}}_i^L \rightarrow x_L \overline{\mathrm{S}}_{i+1}^L) + \mathcal{S}(\overline{\mathrm{S}}_i^R \rightarrow x_R \overline{\mathrm{S}}_{i-1}^R) \geq \mathcal{S}(\mathrm{S}_i \rightarrow x_L \mathrm{S}_{i+1} x_R)$. Since the score of a parse is simply the sum of the scores of the rules used in the parse, clearly the inequalities will enforce the property that $\mathcal{S}(\pi^{\mathrm{HMM}}) \geq \mathcal{S}(\pi^{\mathrm{CM}})$, as required.

4.4.2 HMM vs. CM speed difference

Asymptotically, an HMM with M states can parse an N-nucleotide sequence database in $O(MN)$ time. A CM, in the worst case, requires $O(MNWB)$ time, where W is the CM's window length (see chapter 3) and B is the number of bifurcation states (rules like $\mathrm{S}_i \rightarrow \mathrm{S}_j \mathrm{S}_k$, explained later). The W factor arises because the CM must model subsequences' lengths explicitly (intuitively, to match up base pairs).

However, bifurcation states are needed only rarely (e.g., 0-5 bifurcation states even for moderately large RNAs with a thousand total states). Therefore, in practice, they are not a significant factor in run time, which effectively becomes $O(MNW)$.

So, CMs are roughly $O(W)$ slower than HMMs. Possible constant-factor causes of differences in speed are the lower connectivity in the HMM, the extra expense in the CM for bifurcation states and miscellaneous differences in computer programs implementing CM and HMM scans.

For expanded-type profile HMMs, the average speedup of HMMs over CMs is 1.09 times that of the window length, e.g., if the window length is 100, then the HMM is expected to be 109 times faster than the CM. The range is narrow—from 0.76 to 2.34—which suggests that the window length parameter accounts for most of the speed gain in HMMs.

4.4.3 Filtering with the profile HMM details

Recall that in the standard covariance model framework (see chapter 3), a parameter must be set by the user defining the most nucleotides that a member of a particular family can be. This parameter is the *window length.*

As noted earlier in this chapter, the profile HMM assigns a score to all subsequences ending at a given nucleotide position in the database sequence. If one of these scores is above the family-specific threshold, then there may be a family member ending at that nucleotide position. If the window length for the family is W and i is the nucleotide position where the threshold score was exceeded, then there might be a homolog in the interval $i - W + 1, i - W + 2, \ldots, i - 1, i$.

Thus, each time the profile HMM reports a score above the threshold, there is an interval that must be scanned. To save time, overlapping intervals are merged into larger intervals, and then scanned. In practice, during the HMM Viterbi algorithm, each new score we obtain is for a nucleotide to the right of previously scored nucleotides, so the new interval either partially overlaps the previous interval, or does not; the merging of intervals is thus simple.

4.4.4 Infinite-length forward algorithm details

The optimization procedure proposed earlier in this chapter involves an objective function (the infinite-length forward algorithm) and constraints (designed to enforce the rigorous property). To solve these problems, I used CFSQP [77], a general non-linear program solver. These solvers require a user function to evaluate the objective function and to calculate partial derivatives. These derivatives are required to guide optimization, and without them, only less efficient solvers could be used. The function and its derivatives must be evaluated several times during optimization, so speed is important. I now show how to evaluate the infinite-length forward algorithm and its derivatives, and how to do so efficiently.

Mathematical derivation

In words, the heuristic developed earlier in this chapter is to use the expected HMM forward algorithm result over subsequences of unbounded length, where the subsequences are generated from a 0th-order model.

I now formalize this idea mathematically, and show that the forward algorithm result can be calculated efficiently and derivatives taken. Let π represent an HMM parse, where the parse emits the subsequence x^π. Let $P(\pi)$ be the product of the odds ratios of rules in parse π; the forward algorithm will sum $P(\pi)$ over all π that are consistent with a given database subsequence. Since we are calculating an expected sum for 0th-order subsequences, the expected sum E is

$$E = \sum_{\pi} P(\pi) \Pr(x^\pi) = \sum_{\pi} P(\pi) \prod_{k=1}^{|x^\pi|} \Pr(x_k^\pi)$$

where x_k^π is the kth nucleotide of x^π, $\Pr(x_k^\pi)$ is its probability according to the 0th-order model, and $|x^\pi|$ is the length of x^π.

We can efficiently solve this formula with dynamic programming, computing

$$T_{\overline{\mathrm{S}}_i} = \sum_{\pi(\overline{\mathrm{S}}_i)} P(\pi(\overline{\mathrm{S}}_i)) \prod_{k=1}^{\left|x^{\pi(\overline{\mathrm{S}}_i)}\right|} \Pr(x_k^{\pi(\overline{\mathrm{S}}_i)})$$

for all (left or right) HMM states $\overline{\mathrm{S}}_i$ where $\pi(\overline{\mathrm{S}}_i)$ is a parse prefix ending at state $\overline{\mathrm{S}}_i$. Thus, $E = T_{\overline{\mathrm{S}}_e}$ for HMM end state $\overline{\mathrm{S}}_e$. Let $R_{\overline{\mathrm{S}}_i}$ be the rules with state $\overline{\mathrm{S}}_i$ in the right side, and let $P(\overline{\mathrm{S}}_{i\pm1} \rightarrow x_L \overline{\mathrm{S}}_i)$ be the odds ratio of a rule in $R_{\overline{\mathrm{S}}_i}$ (±1 in $\overline{\mathrm{S}}_{i\pm1}$ because $\overline{\mathrm{S}}_i$ can be left or right).

I note that the forward algorithm does not use logarithms, so the logarithmic score variables must be exponentiated, e.g., $P(\overline{\mathrm{S}}_3^L \rightarrow u\overline{\mathrm{S}}_4^L) = 2^{l_1}$, where l_1 is a logarithmic score variable like those used in the linear inequalities. I also note that even the exponentiated scores are not real probabilities (i.e., they do not sum to 1), but just upper bounds on odds ratios; it may seem strange to use the forward algorithm on numbers that do not really correspond to probabilities. However, empirically it works well, at least for the non-local CMs.

More theoretically, the forward algorithm is an upper bound on the Viterbi (if we exponentiate scores before using the Viterbi algorithm); the Viterbi parse is one that the forward algorithm sums over, but the forward algorithm will typically add the (pseudo) probabilities of other parses. It is generally thought that this bound is reasonably tight since the probability of sub-optimal parses decays exponentially. Therefore, the non-Viterbi parses will usually contribute little to the forward algorithm's result. Since the exponentiated scores used in the infinite-length forward algorithm are multiplicative (like probabilities), the argument for approximation matches that of the probabilistic case.

When non-Viterbi parses contribute more significantly, they obviously must have comparable scores to the Viterbi's. Their impact on the infinite-length forward algorithm may be less significant than for the forward algorithm in its normal usage because the infinite-length forward algorithm is intended to estimate an HMM's filtering performance on a variety of random sequences. If many non-Viterbi parses have scores comparable to the Viterbi's, then those other parses are likely to be Viterbi for some other sequence; counting these other parses may therefore not lead to so much overall error.

Most importantly, the infinite-length forward algorithm produces selective (but rigorous) HMMs in the non-local case. As noted previously, for local CMs (see chapter 6) it was necessary to create a different heuristic, since the infinite-length forward algorithm has serious problems in the local case. For the non-local case the other algorithm is marginally better, but takes much longer to compute (typically 8-10 CPU hours for most families, versus 1-10 minutes for the infinite-length forward algorithm).

Given the above, I obtain the recurrence:

$$T_{\overline{\mathrm{S}}_i} = \sum_{(\overline{\mathrm{S}}_{i\pm1} \rightarrow x_L \overline{\mathrm{S}}_i) \in R_{\overline{\mathrm{S}}_i}} T_{\overline{\mathrm{S}}_{i\pm1}} P(\overline{\mathrm{S}}_{i\pm1} \rightarrow x_L \overline{\mathrm{S}}_i) \Pr(x_L)$$

where $\Pr(x_L)$ is the 0th-order model's probability of nucleotide x_L, and $\Pr(\epsilon) = 1$. This recurrence allows evaluation of the infinite-length forward algorithm in time linear in the number of HMM states, speeding HMM optimization. ($|R_{\overline{\mathrm{S}}_i}| = O(1)$.)

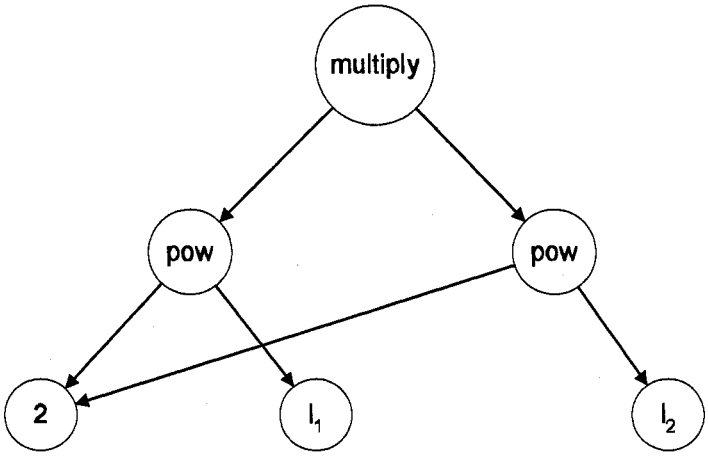


Figure 4.1: Symbolic expression DAG. An expression DAG representing the mathematical expression $2^{l_1} \cdot 2^{l_2}$. The "2" node has two parents.

Partial derivatives of the infinite-length forward algorithm

Gradients are used by many powerful and generic optimization algorithms—in particular, by CFSQP. Gradients of the expected forward algorithm score can be calculated by treating the dynamic programming algorithm symbolically, and differentiating the resulting expression with respect to each of the HMM score variables (e.g., l_1, l_2, l_3 used earlier).

Creation of expression DAG & partial derivatives We need to create a symbolic representation of the infinite-length forward algorithm score and then evaluate the partial derivatives of this function. The symbolic representation I use is a DAG (Directed Acyclic Graph) whose nodes represent mathematical operations (e.g., multiplication); and whose leaf nodes represent numerical constants and logarithmic score variables. The dynamic programming nature of the infinite-length forward algorithm means that the mathematical expression is best represented as a DAG, since there are frequent common subexpressions.

Figure 4.1 shows a DAG corresponding to the mathematical expression $2^{l_1} \cdot 2^{l_2}$. Instead of writing a function (in some computer programming language) to compute the numerical result of the expression given numerical values of l_1 and l_2, we could instead create a DAG representing the expression. For example, the multiplication operator normally

takes two numbers and returns their product, but could instead take two DAG symbolic expressions, and return a DAG representing the product of the input DAG expressions. (In my implementation, I wrote a function that computes the infinite-length forward algorithm using a parameterized "Number" type that was a C++ template parameter. I then replace this abstract Number type with a class that builds expression DAGs instead of evaluating expressions numerically.)

Given an expression DAG, we can evaluate the function. One annoyance is that, since the expression is a DAG, we cannot simply traverse it as if it were a tree, since we would explore nodes multiple times and the evaluation would take exponential time. By storing an "is visited" flag at each node and caching the computed subexpression value, we can avoid this problem.

To evaluate partial derivatives, I use rules from basic calculus. For example, suppose n is some node in the expression DAG and it is an addition node, with operand nodes (its children) f and g; i.e. the value of n will be the sum of the values of f and g. If l_i is some logarithmic score variable, then $\frac{\mathrm{d}n}{\mathrm{d}l_i} = \frac{\mathrm{d}f}{\mathrm{d}l_i} + \frac{\mathrm{d}g}{\mathrm{d}l_i}$. If n is a multiplication node, I use the product rule. Using basic calculus we can derive rules like these to recursively evaluate partial derivatives from a DAG.

Optimizing the expression DAG by combining like terms & constant folding Since it is too computationally expensive to optimize the entire profile HMM at once, I only optimize one CM state at a time. In particular, I optimize the HMM score variables corresponding to a given CM state while holding all other HMM score variables as fixed constants. Therefore, the resulting expression DAG will be large, but will contain many constants. This expression DAG and its partial derivatives will need to be evaluated multiple times during the search for the optimal scores; this process would be faster if we could reduce the size of the DAG by evaluating any constant expressions and combining like terms. For example, given an expression DAG corresponding to the expression $5 \cdot 2^{l_1} + 8 \cdot 2^{l_1}$, we would like to simplify to the term $13 \cdot 2^{l_1}$.

My solution is somewhat general, but was only intended to solve the problem for expressions generated by the infinite-length forward algorithm. I create a special node type to

help to combine like expressions. This "linear function" node type has k children $n_1, \ldots, n_k$ for some k, and its value is given by $\sum_{i=1}^{k} c_i n_i$, where n_i is the ith child node and c_i is a constant real number (i.e., it is a linear function of the values of its child nodes.)

When two linear function nodes are added together, any child nodes they have in common (i.e., like terms) are easily detected and their constants (c_i) added together. If a multiplication node is added to a linear function node, and one of the terms in the multiplication node is a constant, it can be added as a child node of a linear function node. (If the non-constant child of the multiplication node is a linear function node, we can add it in with the multiplied constant.) In this manner, we can simplify the expression DAG as it is being built based on the infinite-length forward algorithm. And, of course, if any constants are added or multiplied, it is easy to detect this, and replace the sub-expression with its value. These tactics significantly reduce the complexity of the resulting expression DAG.

4.4.5 Details on optimizing HMMs

To create an optimized profile HMM-based rigorous filter, we minimize the infinite-length forward algorithm score subject to the linear inequality constraints. This section gives some additional technical details on this process.

A convex optimization problem

The non-linear optimization problem defined by this strategy is a kind of convex optimization problem. The constraints are linear and therefore convex. The infinite-length forward algorithm objective function, if fully expanded, is the sum of terms, where each term is the product of some positive constant and 2 raised to the sum of some subset of variables, e.g., $3 \cdot 2^{l_1+l_2} + 7 \cdot 2^{l_1+l_3}$. The constant coefficients (like 3 and 7 in this example) are always positive because they are sums and products of upper bounds on odds ratios, and odds ratios must be positive. Since the exponents are linear (always a sum of variables), they are convex. The exponentiation of a convex function is always convex, and the positive weighted sum of convex functions is convex.

Start with a linear program

To create a good feasible point to start the optimization procedure, we first solve a linear program for each CM state. The linear program uses a linear objective function instead of the infinite-length forward algorithm. This linear objective function is the sum of the slack variables for each of the linear inequalities. Although the infinite-length forward algorithm is much better at picking scores than this simple linear function, starting with the scores given by the linear program may reduce convergence time.

These linear programs can be underconstrained, allowing solutions that do not help convergence[3]. An alternate scheme is to set each variable to $+10$, which will in practice be a feasible starting point, and run the convex optimization from that point. This increases convergence time, but not fatally, given that scanning time is so much greater. More aggressive and robust schemes for a feasible starting point are also possible.

Questioning optimizing each CM state in isolation

We optimize the HMM score variables corresponding to each CM state in isolation, treating all other HMM score variables as fixed constants; but, perhaps the results would be better if we optimized all HMM score variables at once. We also create linear inequalities that are specific to each CM state; but, perhaps the results would be better if we had created the exponentially many inequalities, one for each valid CM parse.

Intuitively, we expect that treating each CM state in isolation is not limiting the filtering fraction of the resulting profile HMMs because (1) convex optimization problems do not have any local minima to get stuck in, so the global infinite-length forward algorithm is not likely to enable better convergence than the CM state-specific version, and (2) even if one creates linear inequalities spanning two adjacent CM states (instead of considering each CM state in isolation), these inequalities will essentially be the cross product of inequalities that were specific to each individual CM state, because of the repetitive structure of CMs; so, the more general inequalities do not seem likely to actually expand the feasible region of the convex optimization problem, and therefore should not yield any freedom to select more

[3]Thanks to Diana Kolbe for pointing this out

optimal scores.

To explore this empirically, I tried to optimize all variables simultaneously (as opposed to each state in isolation) for RF00008 and RF00029 in Rfam 4.1 with the CM state-specific linear inequalities. This improved the infinite-length forward algorithm score of the optimized HMM by only a very marginal amount. Where the logarithm of the infinite-length forward algorithm in an optimized HMM is typically on the order of 10, I observed improvements of at most 0.008, which may be simply the result of numerical issues.

Although I could not test the use of the exponential number of linear inequalities corresponding to full HMM parses, I tried generating linear inequalities grouping pairs of two adjacent CM states together, instead of only doing one CM state at a time. This did not improve results at all.

I therefore conclude that the strategy of consider CM states in isolation does not hurt the filtering fraction of the produced profile HMMs.

4.4.6 Fully general CMs & compact- vs. expanded-type HMMs

Up to this point, I have considered only simplified CMs. In this section, I first show how to handle CMs with bifurcation and insertion states, then show how to handle fully general CMs as they have been presented in the literature. Finally, I explain the difference between compact- and expanded-type HMMs.

Covariance models with bifurcation and insertion states

RNAs with multiple hairpins are handled by CM bifurcation states, which permit rules like $S_i \rightarrow S_j S_k$ for $j, k > i$, where S_j and S_k will each become a substructure. Bifurcations can be handled easily. The sub-CMs rooted at S_j and S_k are converted to HMMs. The HMM grammars are then concatenated and substituted for the bifurcation state S_i.

My technique also works with insertions relative to the consensus alignment. CMs allow insertions using extra states IL_i and IR_i. Any number of nucleotides can be inserted on the left with rules like $IL_i \rightarrow x_L IL_i$, or on the right with $IR_i \rightarrow IR_i x_R$ $(x_L, x_R \neq \epsilon)$. Any state S_i may transition to an IL_i state with rule $S_i \rightarrow x_L IL_{i+1} x_R$. Ending the insertion

is permitted by $\mathrm{IL}_i \rightarrow x_L \mathrm{S}_{i+1}$, transitioning to the next alignment position. Getting into IR_i states is allowed directly, by $\mathrm{S}_i \rightarrow x_L \mathrm{IR}_{i+1} x_R$, or after a series of left insertions, by $\mathrm{IL}_i \rightarrow x_L \mathrm{IR}_i$. Transitioning to the next alignment position is allowed by $\mathrm{IR}_i \rightarrow \mathrm{S}_{i+1} x_R$.

Each CM state IL_i corresponds to HMM state $\bar{\mathrm{I}}_i^L$ (no right HMM state is needed). Similarly IR_i maps to $\bar{\mathrm{I}}_i^R$. A rule $\mathrm{S}_i \rightarrow x_L \mathrm{IL}_i x_R$ becomes three HMM rules. The left rule is $\bar{\mathrm{S}}_i^L \rightarrow x_L \bar{\mathrm{I}}_i^L$. The right side is more difficult, since CM state IL_i can transition to a right insert IR_i (if any) or the next position S_i. So I add two HMM rules: $\bar{\mathrm{S}}_i^R \rightarrow x_R \bar{\mathrm{I}}_i^R$ and $\bar{\mathrm{S}}_i^R \rightarrow x_R \bar{\mathrm{S}}_{i-1}^R$ (x_R is from the rule $\mathrm{S}_i \rightarrow x_L \mathrm{IL}_i x_R$ here.). The insert state self loop rule $\mathrm{IL}_i \rightarrow x_L \mathrm{IL}_i$ becomes HMM rule $\bar{\mathrm{I}}_i^L \rightarrow x_L \bar{\mathrm{I}}_i^L$. The exit rule $\mathrm{IL}_i \rightarrow \mathrm{S}_{i+1}$ becomes $\bar{\mathrm{I}}_i^L \rightarrow \bar{\mathrm{S}}_{i+1}^L$. The construction for right inserts is analogous.

To create the inequalities, we must consider sub-parses that begin at CM state S_i and end at S_{i+1}, where these sub-parses can contain multiple rules and transition through an insert state. A problem is that insert states have self-loops, e.g., $\mathrm{IL}_i \rightarrow x_L \mathrm{IL}_i$, so the number of sub-parses is infinite. Fortunately, we can set the HMM self-loop scores equal to the CM self-loop score, e.g., the score of HMM rule $\bar{\mathrm{I}}_i^L \rightarrow x_L \bar{\mathrm{I}}_i^L$ will be equal to CM rule $\mathrm{IL}_i \rightarrow x_L \mathrm{IL}_i$. Since $\mathrm{IL}_i \rightarrow x_L \mathrm{IL}_i$ always corresponds to HMM rule $\bar{\mathrm{I}}_i^L \rightarrow x_L \bar{\mathrm{I}}_i^L$, the two scores will cancel out on either side of the inequalities.

An HMM insert state's self loop slightly complicates the infinite-length forward algorithm. Each time the self loop is visited, this multiplies the odds ratio by some number. The loop may be visited 0 or more times, so the total multiple is the sum of an infinite geometric series. As above, the HMM self loop score is a constant, equal to the corresponding CM self loop score, and in practice the logarithmic score is less than 0, so the infinite sum is finite.

My technique in conventional CM terms

This section is aimed at readers who are familiar with the technical details of CMs as they are presented in previous work by other authors, e.g., [27, chpt. 10] or [33]. Other readers may find this section difficult to follow.

Readers familiar with CMs will notice that I have explained them in an unusual way. Moreover, the pedagogical version of CMs is not quite as powerful as true CMs. In this

section, I show how my algorithm fully supports CMs as they are presented in the literature. In fact, RAVENNA, the computer program I wrote to perform the experiments, uses the source code of the Infernal package (`infernal.wustl.edu`), which implements CM searches for the Rfam database.

The form of SCFGs I described are more restrictive than CMs in that the S_i states would have to take the place of the MP,ML,MR,D,S and E states of CMs. The separation of these states in true CMs allows more flexibility in the scores, since scores can be conditioned on which of the types of states, MP, ML, etc., was last visited.

However, the algorithm as presented is straightforward to extend to CMs. The reader will note that the CM states S_i, IL_i and IR_i together correspond to one CM node for each i. The S_i state corresponds to one of the *split set states* [31] of the node, usually MP,ML,MR,D. In the same way that the S_i state must be visited exactly once, exactly one of the CM split set states must be visited exactly once.

Consider a MATP node. This will correspond to a left HMM node and a right HMM node — $\overline{S}_i^L$, $\overline{S}_i^R$, $\overline{I}_i^L$ and $\overline{I}_i^R$ in my earlier explanation. The $\overline{S}_i^L$ state must be split into two states, corresponding to ML and D types, and the same for the $\overline{S}_i^R$ state. The use of the MP state in the CM node corresponds to the use of the ML state in both left and right HMM nodes. The ML state in the CM is an ML state in the left HMM node and a D state in the right HMM node, etc.

For a MATL node, which has no MP, MR or D states, the right HMM node can be a dummy node, with an analogous situation for MATR nodes. The dummy node consists of a single state which transitions to the next node. ROOT, BEGL and BEGR nodes do not require separate split set states, but can use HMM nodes of type D (which emits nothing) in both left and right HMM nodes. Thus, constructing the HMM from a conventional CM is exactly as presented earlier, except that we need to create ML and D states instead of simply a $\overline{S}_i^L$ or $\overline{S}_i^R$ state.

In making constraints for the HMM scores, I considered sub-parses. In terms of a normal CM, a sub-parse corresponds to a sub-path starting at a split set state of one node, and ending at a split set state of the next node. Although there is more than one split set state in a typical CM node, it is still easy to enumerate such sub-paths, find corresponding HMM

Table 4.4: Profile HMM states for a CM MATP node. For each state in a CM MATP node, the corresponding left and right HMM states are shown, for both compact- and expanded-type HMMs. Each expanded-type HMM node (both left and right) has 2 ML states, ML1 and ML2.

CM MATP	**compact-type HMM states**		**expanded-type HMM states**	
node state	**left node**	**right node**	**left node**	**right node**
MP	ML	ML	ML1	ML1
ML	ML	D	ML2	D
MR	D	ML	D	ML2
D	D	D	D	D
IL	IL	-	IL	-
IR	-	IR	-	IR

sub-paths, and create one linear inequality for each sub-path.

Finally, the dynamic programming solution to the infinite-length forward algorithm is sufficient, since it applies as written to any HMM that has no cycles other than self-loops.

Compact- vs. expanded-type HMMs Some flexibility exists in how many HMM left and right states are used to represent a CM's MATP node, and this underlies the difference between compact-type and expanded-type HMMs. In compact-type HMMs, each left and right node has 3 states: ML, D and IL (as described above).

In the expanded-type HMMs, each left and right node has 4 states: 2 MLs, D and IL. One ML state is specifically used to handle the CM's MP state; this separates penalized (non-consensus) states ML, MR and D, from the consensus state, MP. Table 4.4 summarizes the HMM states in a CM MATP node.

Why are expanded-type HMMs more powerful? Empirically, I have found that expanded-type HMMs are significantly more powerful than compact-type HMMs, in the sense that they yield lower filtering fractions. However, it may seem that the extra states

in expanded-type HMMs are redundant and therefore should not be expected to improve filtering power.

The usefulness of these extra states stems from differences in scores between consensus states, like the MP state in a MATP node, versus non-consensus states, like the ML, MR and D states in a MATP node. From a consensus state, typically the transition to the consensus state in the next node has a relatively high score, while transitions to non-consensus states are severely penalized. Again typically, from a non-consensus state, all transitions tend to have significant but lower penalties. Therefore, once a non-consensus state is entered, it is not so disadvantageous to enter a non-consensus state in the next node.

To see how this relates to expanded-type HMMs, consider a CM with two consecutive MATP nodes, and suppose the HMM Viterbi algorithm is processing the profile HMM's left node corresponding to the first CM MATP node. Perhaps a desirable sequence of emissions/transitions is to emit a (left) nucleotide corresponding to the first MATP node, but then emit nothing (a deletion on the left) for the next MATP node. I now consider how accurately a compact- or expanded-type profile HMM can simulate what the CM would do.

With a compact-type HMM, it is possible to suppose that the CM's MP state emitted the left nucleotide with very high score. Then a transition to a D state in the next CM node would achieve the effect of not emitting another nucleotide. However, due to the score inequalities in a compact-type HMM, this transition to the D state would have the score of a transition from an ML state (instead of the MP state) to the D state. The ML state, like an MP state, also emits a left nucleotide, but an ML to D transition has higher score. This cheating—entering an MP state but leaving an ML state in the virtual CM path—is possible with compact-type HMMs since the MP and ML states in the CM correspond to the same left HMM state. The compact-type HMM cannot prevent this inconsistency.

By contrast, an expanded-type HMM has separate states for MP and ML in the left HMM node, and therefore this kind of cheating is not possible. By enforcing this more stringent requirement, the expanded-type HMM therefore makes it more difficult to obtain high scores, and so achieves a tighter upper bound on CM scores.

Proof of rigorousness with full CMs The proof of rigorousness given earlier can be extended to full CMs. In this case, instead of decomposing CM and HMM parses into rules, we must decompose them into sub-paths, i.e. a sequence of rules starting at one CM node and ending at the next. It is clear from the construction of inequalities that the score of each HMM sub-path is an upper bound on that of the CM sub-path to which it corresponds. This can be verified by inspection, given that IL and IR self-loops are set to the same score in HMM and CM, and therefore cancel out immediately. Bifurcation states are easily handled since the bifurcation itself corresponds to a CM sub-path with one rule of score zero, while the corresponding trivial profile HMM sub-path also has score zero.

To address a possible confusion, I emphasize that the proof of rigorousness only requires that every CM path has a corresponding profile HMM path; the reverse condition is not required. In fact, expanded-type HMMs will have many paths that do not relate to any valid CM path, e.g. if the ML1 state is used for the left while the D or ML2 state is used for the right. Such invalid paths do not affect rigorousness, and can only increase the score above that of the profile HMM path considered in the proof.

Chapter 5

RIGOROUS FILTERING USING LIMITED SECONDARY STRUCTURE

The previous chapter created rigorous filters based on profile HMMs. Surprisingly, although based on sequence conservation alone, these filters enabled rigorous scans for 126 of the 139 ncRNA families in Rfam 5.0 (ignoring those families using local CMs; these are covered in chapter 6). Rigorous filtering remained impractical for the other 13 families.

These 13 families tend to exhibit relatively low sequence conservation, but strong conservation of secondary structure. Not only are these families difficult for our previous rigorous filters, but one would expect that the BLAST-heuristic would miss many homologs in these families, given its reliance on sequence information. Thus, incorporation of secondary structure information is essential for successful filtering of increasingly diverged ncRNA families.

In this chapter, I introduce three innovations aimed at extending practical rigorous filtering to biologically important situations such as these. First, I present two techniques that deviate from the usual profile HMM architecture to include limited structure information that improves filtering at the cost of increased CPU time, while still being rigorous. The *sub-CM* technique mixes CMs and profile HMMs, using CMs for key structural elements. The *store-pair* technique uses additional HMM states to store information to better model key base pairs.

By varying parameters, both techniques can generate many filters; some run very quickly, but may not filter selectively, while others are more selective but slower. My third innovation, to minimize overall scan time, is to run several filters in series, starting with the quickest, and ending with the most selective. I solve the problem of selecting the optimal series of filters as a classic shortest path problem [22].

I applied these techniques to the 13 ncRNA families that were not practical with the techniques of the previous chapter, including biologically important ncRNAs such as the

tRNAs, signal recognition particle (SRP), RNase P, two snRNAs and three riboswitches [135, 125]—mRNA structural elements that regulate thiamin, lysine and vitamin B12 genes. Rigorous scans of the RFAMSEQ database run in practical time for all but 2 of these families. In all 11 successful families, these scans find new hits (putative homologs) missed by the BLAST heuristic; in many cases the new hits are supported by annotations or are otherwise biologically plausible.

tRNAscan-SE is a popular program that identifies tRNAs in sequence databases, and is based on CMs. (See chapter 2.) When applied to tRNAscan-SE, my techniques permit a rigorous scan of three of its four CMs (archaeal, eubacterial and nuclear eukaryotic), finding several hits missed by tRNAscan-SE's heuristics. I was unable to improve speed for its fourth model (organellar), although tRNAscan-SE by default runs the raw CM for these tiny genomes.

Additional technical detail not critical to the main ideas here is in section 5.4

5.1 Results

5.1.1 Summary

Table 5.1 lists results on 11 Rfam ncRNA families and 3 tRNAscan-SE models. Rigorous filtering is practical on these families, averaging about 100 times faster than a raw CM, and for all families uncovers family members missed by Rfam/tRNAscan-SE heuristics.

The rigorous filters run slower than tRNAscan-SE, by a factor of 10 for prokaryotes and 100 for eukaryotes, for which tRNAscan-SE was chiefly designed. However, our scans are rigorous, and the technology is, unlike tRNAscan-SE, not specific to tRNAs. It may be possible to tune tRNAscan-SE to find the new homologs, but for BLAST, this is not straightforward. For example, we ran a BLAST filter for RF00168. Finding even 9 of the 11 new hits required an E-value threshold of 10000, degrading BLAST's selectivity so that searching takes 3 times longer than our rigorous scan. More comprehensive examples of BLAST's shortcomings are explored in the context of heuristic filters, in chapter 7.

Table 5.1: Results of rigorous filtering experiments. Each row in this table is one Rfam or tRNAscan-SE ncRNA family, described in the first column, with Rfam Id if applicable. Next is its average length in nucleotides and % identity (sequence conservation) as reported in Rfam (not available for tRNAscan-SE models). # known is the number of members in RFAMSEQ reported by Rfam, or by running tRNAscan-SE on the appropriate subset of RFAMSEQ. # new is the number of *additional* matches in RFAMSEQ that our technique found. The CPU time taken to scan RFAMSEQ or subset on a 2.8 GHz Pentium 4 is next, then the estimated time for a pure CM scan (extrapolated from a 10 Kbase scan). The RFAMSEQ or subset size is given in megabases. tRNAscan-SE was run with default parameters, with the domain of life specified with appropriate flags; for raw CM and rigorous scans, its default window length of 250 was used. Note: RF00005's scan covered more sequence data, including organellar DNA, so has many more hits than tRNAscan-SE model scans.

ncRNA family	**avg len**	**% id**	**# known**	**# new**	**filters +CM time (days)**	**est. CM time (days)**	**data-base size (Mb)**
RF00001 5S rRNA	115	61	5460	14	14.0	651	8295
RF00004 U2 snRNA	171	60	466	1	9.1	2110	8295
RF00005 tRNA	71	43	58609	5158	31.0	335	8295
RF00009 nuclear RNase P	290	41	69	3	14.2	6240	8295
RF00010 bacterial RNase P	317	62	413	1	31.9	9029	8295
RF00017 SRP	299	49	128	13	14.9	5493	8295
RF00023 tmRNA	345	46	226	21	18.7	9587	8295
RF00029 Group II intron	75	55	5708	331	6.9	666	8295
RF00059 thiamin element	104	52	276	7	10.7	2485	8295
RF00168 lysine riboswitch	181	49	60	11	21.2	1724	8295
RF00174 cobalamin riboswitch	202	47	170	7	38.1	5081	8295
tRNAscan-SE archaea	-	-	1016	15	0.1	5	47
tRNAscan-SE eubacteria	-	-	13624	87	1.8	80	640
tRNAscan-SE *Drosophila* nuclear	-	-	296	1	0.7	21	117
tRNAscan-SE *C. elegans* nuclear	-	-	822	16	2.7	18	100
tRNAscan-SE human nuclear	-	-	608	121	26.2	562	3070

5.1.2 Impractical ncRNAs

This chapter's technique did not improve scan time significantly for two Rfam families. The SECIS element (RF00031) is a single long hairpin structure. My techniques work best on RNAs with many hairpins, where spending more time on one hairpin does not inflate the overall ncRNA run time so significantly. I have no clear reason why RF00177, the 5' domain of the small subunit ribosomal RNA, proved impractical.

The technique was also unsuccessful on tRNAscan-SE's organellar tRNA model. The organellar model is based on training sequence from all three domains of life, which may dilute primary sequence features exploited by profile HMMs. Also, organellar tRNAs' score threshold is set lower than normal, so it is harder to prove that a sequence must score below the threshold.

5.1.3 Buried treasures

All families scanned in this chapter revealed hits missed by Rfam/tRNAscan-SE. (This is in contrast to the previous chapter with profile HMM filters, where the BLAST heuristic found all hits for most families; apparently the harder families for rigorous filtering, which typically have much secondary structure information, are also hard for heuristics like BLAST.)

Many new hits found with rigorous filtering are supported by an annotation. 5S rRNA (RF00001) has 8 hits annotated as such, at least 4 based on studies specifically of 5S rRNA. Eukaryotic nuclear RNase P RNA (RF00009) has 3 hits in *Drosophila*, one of which is a partial RNase P sequence identified in a study of eukaryotic RNase P. Bacterial RNase P RNA (RF00010) has one archaeal hit in an annotated partial RNase P sequence. The thiamin element (RF00059) has three new hits upstream of predicted thiamin biosynthesis genes. The cobalamin riboswitch (RF00174) has new hits in the cbiA and cbiX genes, both required for cobalamin synthesis. Of 11 new hits for the lysine riboswitch (RF00168), all are upstream of genes: 6 to lysine-specific permeases, 3 to more general amino acid transporters, 1 to lysA (lysine biosynthesis), and 1 with no annotated function. Preliminary inspection of the huge number of new group II intron (RF00029) and tRNA (RF00005 and tRNAscan-SE) hits identified several with supporting annotation. Thus, rigorous filters have uncovered

homologs missed by heuristics, and will potentially lead to a better understanding of ncRNA families.

5.2 *Augmented filters*

I present two techniques that augment profile HMMs for more selective filtering. The method to create regular (i.e., un-augmented) profile HMMs is in section 4.2. As in the previous chapter, I describe the filters in terms of the simplified model of CMs in chapter 3. However, these filters can be extended to fully general CMs, in a manner analogous to that of the profile HMM scheme of the previous chapter (see section 4.4.6).

I first describe how to select an efficient series of filters from many potential filters, then how to create candidate filters with each technique. The complete process of filter creation and selection has taken from 1 to 50 CPU hours per family.

5.2.1 Selecting a series of filters

Given a set of filters, we wish to select an optimal series of filters to apply. First, each filter is run on a training sequence to estimate its *filtering fraction* and *run time.* The filtering fraction is the fraction of the original database remaining after filtering, which the next filter or CM must be run on. Fractions range from 0 (perfect) to 1 (worst). Run time is CPU time per kilobase.

As an example, Figure 5.1 shows input filters created with the store-pair and sub-CM techniques, and the series of filters selected.

We wish to select a series of filters, preceding the CM, to minimize expected total run time. For example, suppose filter f_1 has filtering fraction 0.25 and run time 1 second per kilobase, and f_2 has fraction 0.01 at 10 s/Kb. If the CM takes 200 s/Kb, running f_2 beforehand takes $10 + 0.01 \times 200 = 12$ s/Kb. Better is running f_1, then f_2, and then the CM, at $1 + 0.25 \times 10 + 0.01 \times 200 = 5.5$ s/Kb.

To formalize this, we make two assumptions: (1) the estimated fractions and run times are accurate and constant, even though in reality they vary by sequence scanned, and (2) a filter's fraction is unaffected by which filters were applied previously. In the above example,

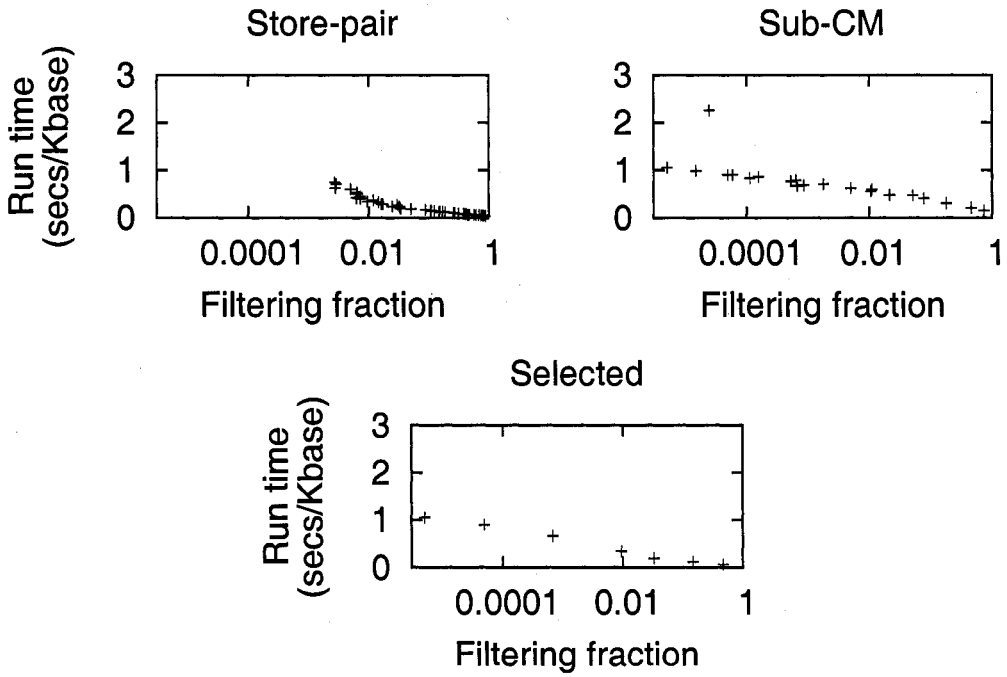


Figure 5.1: Filter creation and selection. Filters for Rfam tRNA (RF00005) generated by the store-pair and sub-CM techniques and those selected for actual filtering are plotted by filtering fraction and run time. The CM runs at 3.5 secs/Kbase. The seven selected filters are run one after another, from highest to lowest fraction.
Among the sub-CM filters, there is one that takes over 2 secs/Kbase, and appears an outlier in the context of the other filters. This is a case where the approximations used in creating sub-CMs are flawed, and lead to a probably useless sub-CM. However, since most filters proposed by the scheme make good speed vs. filtering fraction trade-offs, the outlier is inconsequential.
(Note: this figure uses the automated scheme described in this chapter, so is different from [131], which used a partially manual scheme.)

f_1 followed by f_2 may filter better than f_2's fraction of 0.01, but we have never observed this effect to be significant. If these assumptions are significantly violated, scanning time may be unnecessarily increased, but rigorous filtering is still guaranteed. Additional technical details on this issue appear in section 5.4. (Briefly, test sequences must be sufficient large to obtain acceptably robust estimates, particularly for low filtering fractions. To reduce test scan time for filters with low fractions, large test sequences are prefiltered with less selective filters. The G+C content of the test sequence is also an important consideration.)

Given the above assumptions, selection of the best filter sequence can be cast as a shortest path graph problem [22]. Nodes represent both filters and the CM, and a special INPUT node has fraction 1, taking time 0. The weight of the edge from filter f_1 to f_2 is the time it would take to run filter f_2 immediately after f_1: f_2's time multiplied by f_1's filtering fraction. The shortest path from INPUT to CM yields the optimal series of filters, and is computed in under a second using Dijkstra's algorithm [22].

5.2.2 Sub-CM technique

Sub-CMs exploit information in hairpins, which are base-paired helices (including bulges and internal loops) that end in a single-stranded loop. (See section 1.1.5.) In many ncRNA families, much secondary structure information is carried in short hairpins that use only a fraction of the CM's states. This motivates representing these hairpins using the appropriate part of the original CM (sub-CM), while using a profile HMM for the remainder of the ncRNA. Although the resulting hybrid grammar will take longer to scan than an HMM, it will often filter much more selectively, and still be faster than a full CM.

For a sub-CM rooted at state i, the CM Viterbi algorithm computes the maximum score from $\overline{\mathrm{S}}_i^L$ to $\overline{\mathrm{S}}_i^R$. In the augmented HMM algorithm, the Viterbi score to $\overline{\mathrm{S}}_i^R$ is the highest sum of sub-CM score ($\overline{\mathrm{S}}_i^L$ to $\overline{\mathrm{S}}_i^R$) plus HMM score for state $\overline{\mathrm{S}}_i^L$.

Typical hairpins are much smaller than the window length parameter W of the overall family. Sub-CM scan time could be saved by using a hairpin-specific window length $W' < W$. The lengths $W' \ldots W$ must still be considered, or ncRNAs with unusually long hairpins but otherwise strong homology may be missed. To consider lengths $> W'$ efficiently, the

Table 5.2: Example of converting a CM to a profile HMM. A CM grammar (from Figure 3.1) is converted to a profile HMM grammar, rule by rule. The HMM can be read in sequential order by going down the middle column, then up the right column. Note: the grammar is different from Figure 4.3, to emphasize issues related to store pair.

CM state rules	**left HMM state rules**	**right HMM state rules**
$S_1 \rightarrow cS_2g \mid gS_2c$	$\overline{S}_1^L \rightarrow c\overline{S}_2^L \mid g\overline{S}_2^L$	$\overline{S}_1^R \rightarrow c \mid g$
$S_2 \rightarrow \epsilon S_3\epsilon \mid cS_3\epsilon$	$\overline{S}_2^L \rightarrow \epsilon\overline{S}_3^L \mid c\overline{S}_3^L$	$\overline{S}_2^R \rightarrow \epsilon\overline{S}_1^R$
$S_3 \rightarrow uS_4a \mid uS_4g$	$\overline{S}_3^L \rightarrow u\overline{S}_4^L$	$\overline{S}_3^R \rightarrow a\overline{S}_2^R \mid g\overline{S}_2^R$
$S_4 \rightarrow cS_5\epsilon \mid gS_5\epsilon$	$\overline{S}_4^L \rightarrow c\overline{S}_5^L \mid g\overline{S}_5^L$	$\overline{S}_4^R \rightarrow \epsilon\overline{S}_3^R$
$S_5 \rightarrow \epsilon$	$\overline{S}_5^L \rightarrow \overline{S}_4^R$	

maximum possible score for any sequence of that length is precomputed once and used. If W' is high enough, these maximum scores will be low, and sub-CM filtering will still be selective.

The creation of candidate sub-CM filters is automated. In the interest of readability, the full algorithm is described as part of the complete method of selecting filters in section 5.4.3. The algorithm to select candidate store-pair filters is given in the next section. The store-pair and sub-CM creation algorithms use similar concepts, but the sub-CM algorithm requires more technical details, so is described later.

5.2.3 Store-pair technique

Earlier we noted that profile HMMs cannot reflect which base pair was used, but only which nucleotides were used in the left and right positions. For example, consider the CM and profile HMM in Table 5.2. (This is different from the similar table in the previous chapter.) For position 1, this profile HMM can remember that each side is C or G, but not that only C-G or G-C pairs are allowed.

However, an HMM with extra states can allow only C-G or G-C pairs. For each state $\overline{S}_{1\ldots5}^L$ and $\overline{S}_{5\ldots1}^R$, we create four states, one for each possible nucleotide that could be emitted by $\overline{S}_1^L$. If $\overline{S}_1^L(C)$ is the state for the emission of a C, then rule $\overline{S}_1^L(C) \rightarrow g\overline{S}_2^L(C)$ has score

$-\infty$, since the emission of G is an inconsistency. The rule $\overline{\mathrm{S}}_1^L(C) \rightarrow c\overline{\mathrm{S}}_2^L(C)$ has the same score as in the original profile HMM. Using this extra information on the right side, the rule $\overline{\mathrm{S}}_1^R(C) \rightarrow c$ has score $-\infty$, since the rule is specific to the non-canonical C-C pair. By contrast, $\overline{\mathrm{S}}_1^R(C) \rightarrow g$ corresponds to a C-G pair, and so maintains a high score.

Degrees of freedom

The store-pair strategy can be generalized in three ways. First, any combination of states that represent base pairs in the CM can be multiplied, at the cost of a larger HMM. Note that applying the strategy to base-pair states nearer a loop will result in fewer total states.

To get maximal information, it is desirable to multiply the number of states by 5, which covers the four nucleotides plus the absent nucleotide case, i.e. the 5 symbols $\{a, c, g, u, \epsilon\}$. However, multiplying by 5 is not necessary. For example, in the above example, it suffices to store only 3 possible events: $\{c\}$, $\{g\}$ or $\{a, u, \epsilon\}$—in the last case (not C or G), the scores are the same. (In fact, in my simplified example, the last case leads to score $-\infty$, so does not need to be explicitly stored. A real CM, however, would include all possible base pairs for state S_1, with scores below those of C-G and G-C, so the $\{a, u, \epsilon\}$ case would have to be stored.) More generally, we can choose any partition of $\{a, c, g, u, \epsilon\}$ (i.e., place each of the 5 events into exactly one subset), thus making a trade-off between the improved filtering achieved by having more information versus the increased scanning time of extra states.

Finally, the right nucleotide can be "stored" (i.e. $\overline{\mathrm{S}}_i^R$ instead of $\overline{\mathrm{S}}_i^L$), and used for the left HMM state's scores. This can only make a difference when fewer than 5 events are stored, e.g., the preceding example where the 3 events $\{c\}$, $\{g\}$ and $\{a, u, \epsilon\}$ were stored. Mathematically, this is simply the reverse of what was done above. (Thinking about storing left or right nucleotide as a non-deterministic choice may illustrate the similarity.) When fewer than 5 events are stored, storing the right nucleotide may yield more information than the left.

Creating filters

Exploiting these degrees of freedom in the store-pair technique, I now show how to create a useful set of store-pair filters, obtaining a time versus filtering curve as in Figure 5.1 from which to select a series. Noting that the run time of a store-pair filter is roughly proportional to the number of states in the resulting HMM, I propose to find, for each possible number of states, the filter with the lowest filtering fraction.

To efficiently solve this problem, I make three simplifying assumptions. First, instead of trying to minimize the filtering fraction, I attempt to minimize the average Viterbi score; if the Viterbi score is reduced, then fewer scores should be above the threshold, and the filtering fraction reduced. The second assumption is one of independence: the reduction in Viterbi scores versus the original profile HMM caused by applying store-pair to a set of base pair positions is the sum of the Viterbi score reductions for each individual base pair position. Finally, the user selects a constant c; if the profile HMM has n states, the store-pair HMMs should have fewer than cn states. $c = 250$ is a generous bound, since empirically the HMM is faster than the CM by a factor of approximately W (the window length); see section 4.4.2. If $c > W$, the filter will be slower than the CM.

These assumptions permit a dynamic programming algorithm. For the ith CM state, while restricting store-pair to states $i \ldots n$, we recursively compute the optimal store-pair HMM of each possible number of states from 1 to cn. The optimal HMM and its estimated score reduction are stored by number of HMM states.

The base case, state n, has just the rule $S_n \rightarrow \epsilon$. There is no room to apply store-pair, so the table has only one entry, containing a score reduction of zero. For the recursion, we first enumerate all store-pair modifications of state i. For each modification, we run the resulting HMM on a short training sequence, to calculate its average Viterbi score reduction. We then consider each entry in state $i+1$'s table. The combined score reduction is the sum of the score reduction just estimated on the training example plus the score reduction in the $i+1$ table entry. The resulting number of states can be computed with simple arithmetic. The table for state i is then updated with the new HMM, unless there is already a better HMM with the same number of states. Bifurcation child states' tables are combined analogously.

Finally, we obtain a table for the first state giving, for each number of states up to *cn*, the (heuristically) optimal store-pair HMM. To prune away the large number of similar-performing HMMs, we run through the table, beginning with the fewest states, looking for the first HMM that predicts a Viterbi score reduction of at least 0.3 (a configurable parameter). We then store this HMM in a file, and look for the first HMM with predicted score reduction of an additional 0.3, until the table is exhausted, and a set of HMMs is saved.

I note that, as an alternative to calculating the average Viterbi score on a test sequence, I have been using the logarithm of the infinite-length forward algorithm score, which was proposed in the previous chapter as an approximation to the expected Viterbi score. On Rfam 5.0 tRNA (RF00005), I found that the two statistics correlated (correlation coefficient 0.999), and produced substantially the same filters, but the infinite-length forward algorithm score can be computed more quickly. (As noted previously, this is probably not the case for local CMs; see chapter 6.)

5.3 Discussion

Future work for this chapter (and others) is discussed in chapter 11. A limitation of rigorous filters in this and the previous chapter is that they do not apply to local CMs. This limitation is partially addressed in the next chapter.

5.4 Additional details

This section provides additional information:

- Additional technical details on filters. This section mentions details on the implementation of scanning with sub-CM filters.

- Conceptual issues with filter creation and selection. This section describes issues that the automated method must deal with.

- The automated process for filter series creation and selection.

- How accurate are the approximations used in filter selection? A number of approximations and associated assumptions were used to efficiently select a series of filters; this sections considers how accurate these approximations are in practice.

5.4.1 Additional technical details on filters

Section 5.2.2 says that, in the augmented HMM with sub-CM rooted at state S_i, the Viterbi score to state $\overline{S}_i^R$ is the highest sum of sub-CM score ($\overline{S}_i^L$ to $\overline{S}_i^R$) plus HMM score for state $\overline{S}_i^L$. In fact, the sub-CM (in accordance with the standard CM/SCFG algorithm) finds Viterbi scores from $\overline{S}_i^L$ to $\overline{S}_i^R$ for a given length sequence. (The sub-CM score from $\overline{S}_i^L$ to $\overline{S}_i^R$ is equivalent to the score for the CM rooted at S_i.) If W' is the sub-CM-specific window length, we must consider various lengths, up to W', in stitching together the HMM and sub-CM Viterbi paths.

To find the augmented HMM's Viterbi score to state $\overline{S}_i^R$ at nucleotide position j,

for $w = 0 \ldots W'$ **do**

score $\leftarrow$ (augmented HMM score to state $\overline{S}_i^L$ at nucleotide position $j - w$) + (sub-CM score from state $\overline{S}_i^L$ at position $j - w$ to state $\overline{S}_i^R$ at position j).

best-score $\leftarrow$ max(best-score,score)

end for

To implement this algorithm efficiently, during the augmented HMM Viterbi algorithm, for each state $\overline{S}_i^L$ with sub-CM rooted at state i, the Viterbi score to state $\overline{S}_i^L$ is stored for the most recent W' positions.

5.4.2 Conceptual issues in filter creation and selection

G+C content

G+C content strongly relates to filtering fraction, as shown in Figure 5.2. For some families, filtering is easier with high G+C, while for others it is easier with low G+C. The difference between high and low G+C is dependent on the family. (Intuitively, it depends on the typical G+C content of family members.) For example, for Rfam 5.0 family RF00010, the

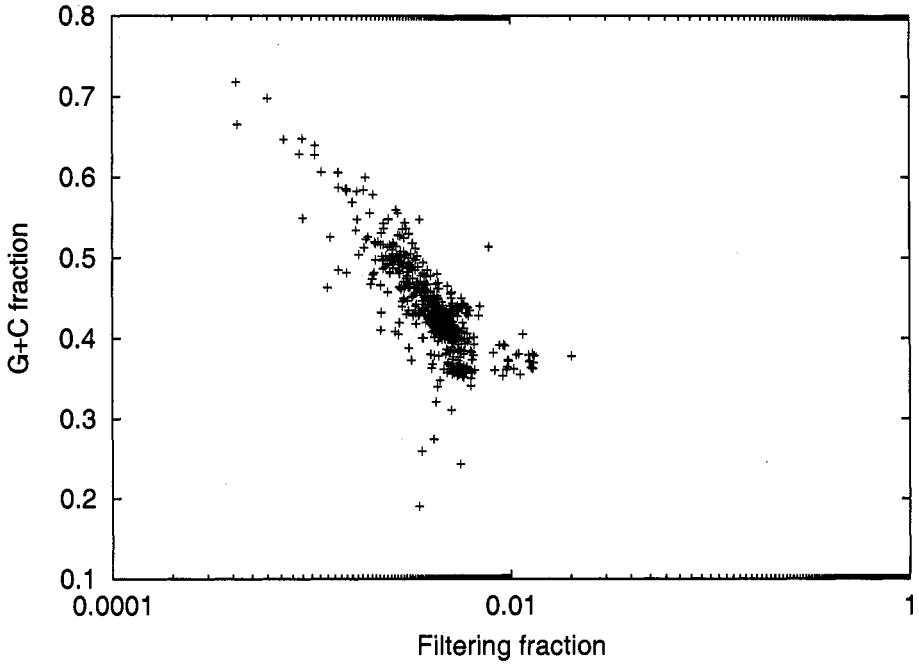


Figure 5.2: Filtering fraction correlates with G+C content. The 8-Gbase RFAMSEQ genome database was divided into 5 Mbase chunks. For each chunk the G+C content was calculated as a fraction of nucleotides, and the filtering fraction was calculated for a profile HMM rigorous filter built from RF00029 in Rfam 4.1. The correlation coefficient is -0.64.

filtering fraction of an expanded-type HMM on the *E. coli* genome is 0.749; on the *S. aureus* genome it is 0.069 (a difference of an order of magnitude). By contrast, for RF00015, the filtering fraction is 0.00148 for *E. coli* and 0.00150 for *S. aureus* (no significant difference at all). The G+C content of *E. coli* is 50.8% versus 32.8% for *S. aureus*.

It is generally reasonable to plan filtering based on sequence data corresponding to the least favorable G+C content, because (1) what is optimal in the worst case is typically close to optimal in easier cases, and (2) since filtering fractions are much higher in the worst case, worst case performance tends to dominate the time taken for a large database scan. (I note that a more sophisticated scheme that selects different filter series depending on G+C content may improve performance somewhat, but can be risky: it is difficult to rule out the possibility that a sequence has small regions of difficult G+C content, and if a more aggressive filter series is selected for this region, it may be extremely inefficient.)

Appropriate test sequence sizes and "pre-filtering"

I now consider what is an appropriate sequence size to test filtering fraction. With a low filtering fraction and a small sequence, the estimated fraction may represent a small number of nucleotides whose upper bound scores exceeded the score threshold. Since these are rare events, the estimates are not very reliable. We must therefore use large sequences in order to obtain reasonable estimates when the filtering fraction is relatively small.

Unfortunately, larger test sequences require more CPU time to scan; to obtain a perfect estimate, we could scan the entire database, but this is clearly not practical if we wish to test many filters whose CPU time requirements may not be that much smaller than the CM's. We need to make intelligent trade-offs between the need for accuracy and the time required to test.

To solve this problem, I use *pre-filtering*. A large test sequence is used, and is pre-filtered once with a less selective filter. The pre-filter has a higher filtering fraction than the filter we wish to test. We then test the filter on the pre-filtered sequence. This pre-filtering strategy allows a more efficient test of highly selective filters, since it only tests them on the harder subsequences. Details on this scheme appear later.

Reliability of CPU timing

To estimate run time, the computer's clock is used to time the filtering process. The logic assumes that this process will give consistent results.

I have found this assumption to be violated on multi-processor machines, where timing a subroutine can give radically different results, with times varying often by a factor of 2. On uniprocessor machines, times seem to be consistent to within a fraction of a second, so I use these uniprocessor machines for filter creation and selection. I have not investigated the cause of the inconsistency on multi-processor machines.

5.4.3 The automated process for filter series creation and selection

This section works in terms of fully general CMs as they are presented in the literature. (See section 4.4.6.)

At the beginning, we wish to rigorously filter some database sequence for a given CM. We are given:

- The database sequence, e.g. the approximately 8-Gigabase RFAMSEQ.
- The CM.
- A score threshold above which hits should be reported.

My technique assumes that the above input data is correct; I do not consider the question of how to create a CM or score threshold or how to select a genome database.

We are also given several parameters, which I describe as they are used. All parameters mentioned have a default value, so the end-user is not required to determine values.

Selection of worst-case G+C content.

All filters are measured against sequences that have the worst-case G+C content. Also, the profile HMMs need to be optimized for a 0th-order Markov model (using the infinite-length forward algorithm); for this 0th-order model, we use the worst-case G+C content. The worst-case G+C content is determined in the following manner.

We consider three possible G+C contents, based on bacterial genomes: *Staphylococcus aureus* MW-2 (very low), *E. coli* K-12 (medium) and *Bordetella bronchiseptica* (very high). We will decide which of these bacterial genomes represents the hard case in the following steps:

1. Create a profile HMM filter based on the CM.

2. Test the profile HMM on each genome.

Create a profile HMM. The first step requires a profile HMM. The parameter `hardGenomeHmmType` says what method to use to build the HMM. The main choices are to build a compact-type or an expanded-type profile HMM. In both cases, the profile HMM is optimized to a uniform 0th-order model.

Another option is to create a profile HMM using the ML-heuristic instead of rigorous profile HMM filters. The advantage of this technique is that the creation process is very fast, although its performance will not correlate as well as the rigorous profile HMM techniques.

Test profile HMM on each genome. To test the profile HMM, there are two options, controlled by the parameter `hardGenomeScoreMethod`. The obvious technique is to run the profile HMM on a test sequence, and use the average Viterbi score as an indicator; the genome with the highest Viterbi score is presumed to have the hardest G+C content. The actual test sequence used can be smaller than the genome. To generate the test sequence, a 2nd-order Markov model is learned from each genome, and a sequence of a user-supplied size is generated. The size is specified in the parameter `hardGenomeViterbiSeqSize`.

The other possibility is to use the infinite-length forward algorithm result (see chapter 4.). For each test genome, a 0th-order model is learned. Then the infinite-length forward algorithm is run on the given test profile HMM using this 0th-order model. The genome with the highest infinite-length forward algorithm result is presumed hardest.

Which method should be used? In practice, all of these techniques tend to give highly similar results. Moreover, when they disagree (e.g., one method indicates that *E. coli* was

hardest, while another indicates *Staphylococcus*), the difference in filtering fractions between the genomes is usually small, so the discrepancy is insignificant.

The advantage of the heuristic profile HMM and the infinite-length forward algorithm is that these are quick to evaluate, whereas other techniques take more time. However, a lot of work follows the selection of the hard genome, so spending more time estimating the hard G+C content will not dominate overall time. Moreover, an error in determining the hard G+C content can result in a poor choice of filters. Therefore, I recommend using the expanded-type rigorous profile HMM and the average Viterbi score. (This is the default.)

For the eukaryotic tRNAscan-SE model, the use of the heuristic profile HMM causes *Bordetella* to be considered the hard genome, whereas in fact *S. aureus* is the hard genome. This is a serious difference since the two genomes are at opposite extremes of G+C content, and *Bordetella* is very easy for this model. So the resulting filters are too aggressive; on genomes like *S. aureus*, these aggressive filters are 25% slower. On *Tetrahymena thermophila*-like sequences, they are a factor of 2 slower. (*Tetrahymena* has an even lower G+C content than *S. aureus*.)

Creation of test sequences

Test sequences, on which we measure the filtering fraction and run time of filters, are generated from a 2nd-order Markov model. (Higher-order models are also possible, but do not seem to change results.) This 2nd-order model is learned from the genome with hardest G+C content, e.g., if it is concluded that *Staphylococcus* has the hardest G+C content, a maximum-likelihood 2nd-order model of that genome is learned, and that model is used to generate sequences of arbitrary size. This is necessary, since some test sequences may be larger than the genome.

Test sequences are of various sizes, depending on the filtering fraction of the filter. Sequences of any size can be generated from the 2nd-order model.

Determining test sequence size

As described previously, the test sequence size must be large enough to give statistically reliable results, e.g., filters with low filtering fractions must be tested on larger sequences. Initially the filter is tested on a short sequence. This short initial test sequence size is given by the parameter `initialTestSeqSize`. The recommended value is 10000, which is dictated by the amount of run time it takes a profile HMM to process that size sequence; this run time must be large enough to get a reasonably accurate reading from the computer's clock.

We then determine if the size was sufficient. Intuitively, if the filtering fraction is the result of a small number of events where nucleotides were above the threshold, then the estimate is unreliable; if it is the result of a relatively large number of events, the estimate is reasonably reliable. Each event where a nucleotide's score is above the threshold results in a window of size *window length* being added to the numerator of the filtering fraction, or less if windows overlap. I estimate the number of events as $\frac{s \times f}{W}$, where s is the sequence size, f is the estimated filtering fraction on this sequence size, and W is the window length.

To determine if the number of events is too low, it is compared to the parameter `minEstAboveThresholdEvents`. The recommended value is 10, which seems to give an acceptable level of robustness in the estimates.

If this test indicates that the sequence is too small, the sequence size is increased by a factor of 10 (hardcoded parameter), and the test is repeated with this larger size. Tests are repeated until a size is found to be sufficient, or until the maximum sequence size is reached (described in next paragraph).

Some filters can be extremely selective, having genuinely very low filtering fractions. For these filters, we would need a very large test sequence, which requires a prohibitive amount of CPU time, even with pre-filters. Moreover, once the filtering fraction is low enough, running the CM immediately after it will not affect overall scan time significantly; therefore, it is not important to accurately determine the filtering fraction in this low range. To avoid wasting CPU time, I impose a maximal test sequence size, and accept the filtering fraction estimate on this maximum size even if the earlier test indicates that it is not robust. The maximum test sequence size is given by the parameter `maxTestSeqSize`, with recommended

value 10,000,000.

Prefilters

As discussed previously, much time can be spent testing selective and slow filters on large sequences (e.g. 1 or 10 Mbases). I therefore pre-filter these sequences once with less selective filters. For example, suppose a highly selective filter must be tested on a 10 Mbase sequence, and call this filter the *test filter*. The 10 Mbase sequence is generated from the 2nd-order model, as above. Then a pre-filter is selected. This pre-filter will be a filter that has already been tested (so its filtering fraction has already been estimated). The 10 Mbase sequence is then scanned using the pre-filter, and the remainder is stored as a list of intervals within the original sequence. This remainder sequence is then scanned with the test filter, and the filtering fraction relative to the original sequence is used (i.e. the test filter's filtering fraction is computed as if the sequence were never pre-filtered). The estimated run time is the number of CPU seconds taken for the scan, divided by the size of the remainder sequence (i.e. the size of the sequence we actually ran the filter on). If another filter must be tested on the 10 Mbase sequence, the remainder sequence after the pre-filter is re-used. (It is somewhat important to re-use test sequences, not only for efficiency, but also because it makes the filtering fractions and run time statistics more comparable, since there is no statistical noise resulting from variations in the test sequence.)

To further reduce CPU time, pre-filters can be chained. Thus, suppose a 100 Kbase sequence is scanned, and filter 1 is used as a pre-filter. Then, a 1 Mbase sequence must be tested. First this 1 Mbase sequence is pre-filtered with filter 1, and then an additional pre-filter, filter 2, is selected and applied. In some cases, no acceptable pre-filter can be found; in this event, we simply do not use a pre-filter.

There are a number of issues that this approach raises. A key assumption being made is that the pre-filter is reasonably correlated with the test filter, in other words, that if a pre-filter is able to eliminate a subsequence rigorously, then the test filter will also eliminate this subsequence. If this is not the case, then the test filter will obtain an unfair advantage from the pre-filter. In the main, this is not a serious problem in practice since (1) filters are

created with related techniques and will all use the same underlying profile HMM, so they are highly correlated in this sense, (2) the pre-filter used will be less selective than the test filter, so it is unlikely to be able to eliminate subsequences that the test filter cannot, and (3) in practice filters will be run in series during actual scans, and a filter like the pre-filter is likely to be run before the test filter, so the testing scenario is actually a realistic one.

To reduce the effects of the pre-filter interfering with the estimate of the test filter's filtering fraction estimate, I require that the two filtering fractions are not too close. As stated previously, only already-tested filters are used as pre-filters; since the candidates for pre-filter have already been tested, we already have an estimate of their filtering fractions. We do not know the test filter's fraction a priori. However, we can compute the maximum filtering fraction of a filter tested on a sequence of a particular size. For example, if we are testing a 1 Mbase sequence, the test filter's fraction cannot be above some threshold fraction, since otherwise it would have been sufficient to have estimated based on the 100 Kbase sequence. By considering in reverse our logic for determining the test sequence size, it is easy to solve for the maximal filtering fraction.

I require that the pre-filter's filtering fraction is not within some constant factor of the maximal possible filtering fraction of the test filter. If this condition is violated for some candidate pre-filter, I reject it as a pre-filter. The constant factor is given by the parameter `minRatioPrefilterToEstTestFilterFraction`, which has recommended value 4. (This value is simply an educated guess as to the ideal value. It is also a typical difference between adjacent filters in series selected by Dijkstra's algorithm.)

In a similar vein, we can apply pre-filters one after another. The candidate new pre-filter's filtering fraction cannot be within some constant factor of the previously applied pre-filter's fraction. If it is, the candidate is rejected. If no pre-filter has been applied for the test sequence, we imagine that there is a pre-filter with fraction equal to 1. The constant factor is given by the parameter `minRatioPrefilterToPrevPrefilterFraction`, which has recommended value 3 (also an educated guess).

The use of pre-filters may also distort the estimate of the run time. This is particularly an issue for sub-CM filters. For sub-CMs, recall that the window length is a factor in the run time. When a sequence has been pre-filtered, it may contain many intervals whose lengths

are equal to the window length or slightly more. This will result in faster scans, because many cells in the CM/sub-CM Viterbi algorithm's dynamic programming table will be out of the bounds of the short interval.

This effect does indeed distort the run time estimates for sub-CM filters, but this distortion is desirable. A pre-filter will only be used for sub-CM filters that are highly selective (low filtering fraction). Therefore, if the sub-CM filter is selected as part of the series of filters, it will be preceded by some other filter. In this context, it will again run fast as a result of having relatively small intervals to scan. So, it is more realistic to estimate the run time of these sub-CM filters after a pre-filter has been applied.

(Alternately, a more realistic performance model could perhaps be designed. I am using filtering fraction to measure the effect of running a filter, but this is not a good model of sub-CM or CM performance, because CMs' efficiency depends on the length of intervals due to the extra dimension in CMs' dynamic programming table. In place of filtering fraction, we could probably store some kind of two-dimensional filtering fraction, or simply store the distribution of interval lengths. Then, a somewhat more realistic model of sub-CM performance would use this statistic instead of filtering fraction to compute the edge weights in the shortest path graph problem used to select an optimal series of filters. However, the relatively simplistic use of filtering fraction seems to provide good estimates, and therefore the extra work for a more realistic model seems unwarranted.)

Finally, I have noted that the pre-filters considered as candidates are those that have already been tested. Therefore, in order to have good candidate pre-filters on hand, it is important to test the filters in order from least selective (highest filtering fraction) to most selective (lowest fraction). Suppose, to the contrary, that we started with the most selective filters first. These filters would require large test sequences (e.g. 10 Mbases), but we would not have any pre-filters to apply at this point, since no filters were tested. Therefore, we would have to test the filters directly on these large sequences, wasting CPU time. To avoid this problem, I am careful to try to test filters from least to most selective.

We can predict which filters will be less or more selective in both the store-pair and sub-CM filter techniques. Both techniques estimate a filter's selectivity in terms of predicted average reduction in Viterbi score; assuming these predictions are accurate, we can easily

order the filters from least to most selective before actually testing their filtering fractions. For straight profile HMM filters (compact or expanded type), we predict that they are less selective than any sub-CM or store-pair filter.

Generation of profile HMM filters

The least selective and fastest filters are the profile HMM-based filters. Once the hard G+C content has been determined, the next step is to create compact- and expanded-type profile HMM filters. These profile HMMs are optimized for the 0th-order model given by the hard G+C content. The profile HMM filters' fraction and run time are estimated as described above.

Below we will generate sub-CM and store-pair filters, both of which augment a profile HMM. In all cases, we use the expanded-type profile HMM generated as a candidate filter as a basis for the sub-CM and store-pair filters.

Are the profile HMM filters sufficient?

Generation of profile HMM filters is significantly faster than the full process of generating a series of filters using the sub-CM and store-pair techniques. Moreover, for many families, profile HMM filters lead to a sufficiently fast scan speed. Therefore, after testing profile HMMs, we consider whether these profile HMMs are good enough.

The expanded-type profile HMM will always filter more selectively than the compact-type profile HMM. If the expanded-type HMM filters well, there is no need for the more sophisticated techniques.

To estimate whether it is possible to improve on the profile HMM as a filter, I use the following observation. For the 139 non-local families in Rfam 5.0, we estimated the CPU time required by the expanded-type profile HMM and the CM to run on 1 Mbase of sequence. The CM's time divided by the HMM's time divided by the window length of the CM is on average 1.09 (smallest observed = 0.76, largest = 2.34). Thus, if the filtering fraction is significantly less than the reciprocal of the window length, the total scan time is dominated by the profile HMM, and improving the filtering fraction will not significantly reduce

overall scan time. In this case, therefore, there is no reason to consider more sophisticated techniques. If the filtering fraction is not this low, then more sophisticated techniques may yet improve overall performance.

I therefore introduce a constant parameter `ratioOfWindowLenRecipricolToEpsilonFilteringFraction`, abbreviated k. If the estimated filtering fraction of a profile HMM is less than $1/(kW)$, where W is the window length of the CM, then the profile HMM is sufficient, and the augmented filters are not considered. In this case, we skip to testing the run time of the CM (see below). I recommend $k = 4$.

Store-pair filters

Next to consider is filters using the store-pair technique. Store-pair filters are created as described in the main part of this chapter. Earlier I suggested using the logarithm of the infinite-length forward algorithm instead of computing the average Viterbi score. The infinite-length forward algorithm is computed relative to the 0th-order model given by the hard G+C content. (In some cases it may be better to use the average Viterbi score, since (1) with careful selection of the test sequence size, computing the average Viterbi score may not inflate overall filter creation and selection time too much, and (2) I have found occasional cases where the average Viterbi score can be more accurate. I have not explored this alternate scheme in depth.)

The main part of this chapter describes a constant $c = 250$, where, if the HMM has n states, only store-pair HMMs of up to cn states should be considered. The rationale is that eventually the store-pair HMM filters will be slower than the CM, and therefore useless. As described previously, the profile HMMs are faster than the CM by a factor of approximately the CM's window length. Also, recall that the run time of a store-pair HMM is roughly proportional to its number of states. Therefore, as a rule of thumb, if the number of states in a store-pair HMM is greater than the original profile HMM by a factor of the window length, it is too slow. I therefore introduce a parameter `maxTotalStatesMultipleRatioToWindowLen`, which I temporarily abbreviate as k. Let W be the window length. Then c is set to $c = kW$. A conservatively high value of k would be $k = 1$, but I recommend $k = 0.3$; when the store-

pair run time gets this close to the run time of the CM, it's not very useful, and these slower filters are time-consuming to test.

After creating store-pair HMMs using the dynamic programming algorithm as described earlier, we scan through the table looking for filters that give an estimated increase in the logarithm of the infinite-length forward algorithm of at least 0.3. This number is given by the parameter `minStorePairScoreIncreaseFromPrevious`, for which I recommend the value 0.3. The dynamic programming algorithm's predictions of which filters should be considered are only approximate, and so trying more filters makes it more likely that a good filter will be found; higher values than 0.3 can skip important candidate filters.

Once the set of candidate store-pair filters is generated, their filtering fraction and run time is estimated on the test sequences.

Sub-CM filters

The creation of candidate sub-CM filters is automated in a manner analogous to that for store-pair filters.

Hairpins. First, we find all hairpins in the CM by inspecting the structure of the CM itself. Within each hairpin, we find all CM nodes corresponding to base-pair positions, at which a sub-CM could be rooted. (There is no reason to root a sub-CM at an unpaired position, since that position has no secondary structure for the CM to analyze.)

A hairpin is defined as a CM node whose descendants contain no bifurcation nodes. (It would be possible to consider sub-CMs that are rooted on top of a bifurcation node, but this complicates the scanning algorithm because there are multiple end nodes; I have not implemented this solution, so these cases are not considered. I expect that using sub-CMs on multiple hairpins is in practice roughly equivalent to rooting a single sub-CM on top of these hairpins.) Within a hairpin, it is a simple matter to enumerate all base-paired positions.

Simple sub-CMs. A simple sub-CM filter is one that contains exactly one sub-CM rooted at one CM node. The next step is to (1) enumerate these simple sub-CMs, (2) find their

optimal window length and (3) estimate their average reduction in Viterbi score and run time, which will be used to choose good candidate sub-CM filters for further testing.

I first describe in more detail the method to find the window length for one CM node. In this section, the variable W is the window length of the CM, while the variable W' is used for a proposed sub-CM-specific window length.

A simple idea to find the optimal W' is a binary search for values in the range $1 \leq W' \leq W$. I now refine this idea. First, we need a test to see if a proposed window length is too small (for the binary search). One possibility is to compare the filtering fraction at some $W' < W$ with the filtering fraction at W. If the filtering fractions are the same, we would assume that the proposed window length W' is not too small, i.e., it does not degrade performance. This is not robust, since sometimes the filtering fraction can be 1 in both cases (i.e., when a simple sub-CM is not sufficient to do any filtering, but combining simple sub-CMs might be). I therefore compare average Viterbi score instead.

Let $\overline{V}(x_i, W')$ be the average Viterbi score for a simple sub-CM filter with sub-CM rooted at CM node x_i, using window length W' for the sub-CM. The average Viterbi score is estimated using a test sequence (generated from the 2nd-order model) whose length is the parameter `avgScoreEstSeqSize`, with recommended size 10000. This number needs to be large enough that any timing inaccuracy is insignificant, or else we could not estimate the sub-CM filter's run time accurately. However, statistical reliability is not as much of a challenge as with estimating filtering fraction, because each nucleotide makes a contribution to the average Viterbi score, whereas, in the case of highly selective filters, most nucleotides are too easy to filter and make little contribution to a filtering fraction estimate.

We also need to compare average Viterbi scores to see if the window length is too small; if $\overline{V}(x_i, W') = \overline{V}(x_i, W)$ with $W' < W$, then W' is an acceptably large window length, since the filter is producing the same results as with the larger W.

I allow a bit of flexibility in raising the average Viterbi score slightly, by some small constant; sometimes, this allows a significantly smaller window length W', speeding searches, and does not affect the filtering fraction significantly. I use a parameter `acceptableViterbiLossForWindowLenReduction`, with recommended value 0.001. I will abbreviate this parameter below as V^{ϵ}.

I now discuss the binary search algorithm to find $\widehat{W'}(x_i)$, i.e., the optimal sub-CM window length W' at one specific base pair position x_i. Initially, we know $L \leq \widehat{W'}(x_i) \leq H$. The initial values of the low and high bounds, L and H, will be specified later; for now, suppose $L = 1$ and $H = W$. It is not important to know $\widehat{W'}(x_i)$ to the exact nucleotide. Moreover as its value is higher, the acceptable margin of error is higher. I assume that the acceptable accuracy is proportional to the value, with ratio given by the parameter `acceptableProportionalWindowLenResolution`, for which I recommend 0.2 as a reasonable value. I abbreviate this parameter as r, for resolution, in the following.

Here is the algorithm:

1: Compute $\overline{V}(x_i, W)$ {This is the average Viterbi score with the largest window length we could use.}
2: **while** $H - L > rL$ **do** {Continue until we know the optimal window length to within an acceptably small range.}
3: $W' \leftarrow \frac{L+H}{2}$
4: **if** $|\overline{V}(x_i, W') - \overline{V}(x_i, W)| \leq V^{\epsilon}$ **then**
5: $H \leftarrow W'$
6: **else**
7: $L \leftarrow W'$
8: **end if**
9: **end while**
10: $\widehat{W'}(x_i) \leftarrow H$ {The upper bound, H, has definitely passed the test as being just as good as W}

A problem with the simple binary search just proposed is as follows. Suppose we have a hairpin with one long helix, and suppose all positions in this helix require a long window length (e.g., there is an insert state within the hairpin that has a high self-loop probability, so it allows very long sequences with relatively high probability). We will then do a binary search to estimate the window length for the first helix position, then do a separate binary search for the next helix position, and so on. Each step is relatively expensive, especially with a large window length, so this process is wasteful.

Observe the following: if x_1 and x_2 are two base pair nodes in a CM and x_1 is an ancestor of x_2, then the ideal window length of x_1 must be at least as big as the ideal window length of x_2. This follows because sequences generated from x_1 contain a subsequence generated from x_2, so the typical sequence size from x_1 must be at least as big as that of x_2. So, if a given window length was too small for x_2, leading to a poor filter, it must also be too small for x_1.

I exploit this fact in a kind of double binary search. Let $x_1, \ldots, x_n$ be base pair positions within a hairpin, with x_i an ancestor of x_{i+1}. Let the window length of the full CM be W, and let the optimal window length of position x_i be $\widehat{W}'(x_i)$. Thus, $\widehat{W}'(x_i) \geq \widehat{W}'(x_{i+1})$, which suggests a more efficient algorithm. The algorithm is as follows, and uses the preceding algorithm for $\widehat{W}'(x_i)$ as a subroutine.

1: Compute $\widehat{W}'(x_1)$ with $L, H = 1, W$. {We know $1 \leq \widehat{W}'(x_i) \leq W$ for all i}
2: Execute W-Recurse$(1 \cdots n; 0 \cdots \widehat{W}'(x_1))$

Procedure W-Recurse$(i \cdots j; L \cdots H)$ is now defined. The parameters i and j define the current range in the hairpin, x_i through x_j. L and H are the current bounds on $\widehat{W}'$: $\widehat{W}'(i) \leq H$ and $\widehat{W}'(j) \geq L$.

1: **if** $i = j$ **then**
2: Stop — nothing to do, since we've already computed $\widehat{W}'(x_i)$.
3: **end if**
4: $k \leftarrow \frac{i+j}{2}$. {Rounding up if $i + j$ is odd.}
5: Compute $\widehat{W}'(x_k)$ with given L, H
6: Execute W-Recurse$(i \cdots k-1; \widehat{W}'(x_k) \cdots H)$
7: Execute W-Recurse$(k \cdots j; L \cdots \widehat{W}'(x_k))$

Selection of candidate sub-CMs filters. I now consider how to estimate which sub-CM filters make the best filtering fraction vs. run time trade-offs, including compound sub-CM filters, i.e. those containing more than one simple sub-CM. The algorithm uses a similar strategy to the algorithm to generate candidate store-pair filters.

First, for each simple sub-CM, we compute its average reduction in Viterbi score relative

to the profile HMM. Since we have already computed its average Viterbi score, the reduction is computed by simply subtracting the profile HMM's average Viterbi score. Also, while exploring optimal window lengths for the simple sub-CMs, we also measure their increase in run time versus the profile HMM; we use this increase in run time to estimate run times of filters. (In the store-pair technique we were able to assume that the run time is proportional to number of states, but in the case of sub-CMs, we have no such convenient rule of thumb—although, of course, it may be practical to create a run time model for the CM Viterbi algorithm.)

We build up all possible compound sub-CM filters one hairpin at a time. Let S be the set of known sub-CM filters.

1: $S \leftarrow \emptyset$ {Initially no sub-CM filters are known}
2: **for** each hairpin H **do**
3: Let S_H be the simple sub-CMs in hairpin H, which were previously enumerated
4: $S \leftarrow S \cup S_H \cup S \times S_H$ {$S \times S_H$ is the cross product of S and S_H}
5: **end for**

If we were to run the preceding algorithm, the size of S is exponential in the number of hairpins. This might be prohibitive for large ncRNAs with many hairpins, so it is desirable to discard redundant members of S as the algorithm progresses. Suppose two sub-CM filters in S have average Viterbi scores that are very close to each other. In this case, we expect that these filters will also have similar filtering fractions. Therefore, we should discard whichever takes more run time. To quantify whether average Viterbi scores are close to one another, we say that average Viterbi scores are close if they are within some constant given by the parameter `subcmCombiningScoreResolution`, with recommended value 0.05. After adding a sub-CM filter to S, we see if it has a close average Viterbi score to some existing member of S, and if so discard a filter. (I note that for many families, the full exponential number of sub-CM filters is not so prohibitive; these sub-CM filters could probably be stored in memory. This is not the case for store-pair filters, since with store-pair the base and exponent of the exponential set are much larger than for sub-CM.)

While combining sub-CMs for each hairpin, we keep track of the estimated average

reduction in Viterbi score, as suggested above. We also keep track of the increase in run time. If compound sub-CM filter s is formed by combining sub-CM filters s_1 and s_2, then the estimated average reduction in Viterbi score of s is the sum of s_1 and s_2. The estimated increase in run time is also the sum of that of s_1 and s_2.

Once we have computed the set S, we run through the filters looking for the next filter that increases the estimate average reduction in Viterbi score by at least some constant. This is exactly the same approach as for the store-pair technique, except the parameter can be different. The parameter is called `minCombinedSubcmScoreIncreaseFromPrevious`, and I recommend 0.5.

Once this set of candidate sub-CM filters is generated, their filtering fraction and run time is estimated on the test sequences.

5.4.4 Stopping tests of store-pair or sub-CM filters

As we test either store-pair or sub-CM filters in order from predicted least to most selective, we may find that some filter's fraction is good enough that it is not worthwhile to explore more filters. The logic here is similar to the test of whether the profile HMMs are good enough.

In testing both candidate store-pair and candidate sub-CM filters, we determine if the filtering fraction is below some threshold, and stop testing further filters if it is. The test used is exactly the same as that for profile HMMs, except that I have created an additional parameter `ratioOfWindowLenRecipricolToEpsilonFilteringFractionForAugmenteds`, on the theory that we may wish to have a different value.

This early-stopping option is important particularly for families that are only slightly too hard for a pure profile HMM to work. In this case, relatively large test sequences must be used early on due to the filters' lower filtering fractions (since the family is not so hard), and it is difficult to find good pre-filters if the profile HMMs themselves are not acceptable as pre-filters, so tests can be quite time-consuming. So, it is useful to stop early.

5.4.5 Estimating CM's run time.

For the same reasons as the sub-CM, the CM's run time should be estimated after some pre-filters, since, in the actual scan, it will be run after some filters.

The CM does not truly have a filtering fraction, so we need to decide what size test sequence to use. I use the largest test sequence size that has been generated; this largest sequence will have pre-filters already set up.

5.4.6 Selection of a series of filters.

At this point we have a set of filters (profile HMM, sub-CM and store-pair) each with estimated filtering fraction and run time, and we have estimated the run time of the CM. We therefore apply Dijkstra's algorithm to the shortest path problem (as described earlier) to select a series of filters.

Caveat on automation

Although the entire process is automated, it is clear that the scheme involves several heuristics for finding a good filtering series while not taking too much CPU time. It is possible that as-yet-unknown ncRNA families may defeat these heuristics, and for example take unnecessarily long to find filters for. It therefore may be helpful for a human to inspect the intermediate results of the algorithms to see if the heuristics may not be working well, particularly if the filter selection process has been taking unusually long. However, this is only a theoretical concern.

5.4.7 How accurate are the approximations used in filter selection

The above automated process to select a series of filters used a number of approximations (simplifications) to reality:

1. I assumed that running a filter f_1 before running a more selective filter f_2 will not affect the filtering fraction of f_2,

2. I assumed that filtering fraction and run time estimates obtained on one test sequence (generated from a 2nd-order model) are accurate predictions for real genomic data that may contain a wide variety of sequences, and

3. I assumed that the total CPU time used by a filter or by the CM could be estimated as the fraction of the database on which it is run, multiplied by its run time (s/Kb) (although in this case, since the assumption is noticeably violated by sub-CM filters and CMs, we used pre-filters to model the effects).

I have not extensively tested these assumptions, but they appear to be reasonable approximations. I report some findings on 3 ncRNA families derived by a recent report on riboswitch-like motifs[5], for which I performed rigorous scans on the RFAMSEQ for Rfam 6.0 (i.e., EMBL release 78). I do not report the statistics on the Rfam scans reported in the main part of this chapter, since those scans did not use the automated process above.

The results are in Table 5.3. Qualitatively, the estimates appear generally quite close to the actual values. Moreover, the estimates may actually be better in practical terms than these results suggest; if the estimates are off by a roughly consistent amount, the approximately best filters will likely still be chosen.

Table 5.3: Estimated vs. actual statistics on rigorous filter series scans. For these tests, I used 3 ncRNA families derived from the original glmS and gcvT families previously published [5]. Rigorous filter series were created for each family using the automated procedure described in this chapter. (Some results described in this chapter used a partially manual process; it was not practical to repeat the CPU-intensive tests, but manual decisions were not very extensive, and the fully automated algorithm gives similar results in all cases tested.) Each filter series was then used to scan the RFAMSEQ subset of EMBL release 78. Each of the filters used has an estimated filtering fraction (estimated on a test sequence). This estimated filtering fraction is given in the table, along with the actual filtering fraction measured on the scan of RFAMSEQ. When a series of filters is selected using Dijkstra's algorithm, the total cost of the shortest path is the predicted run time of the series of filters plus the CM, in secs/Kb. This estimate is given, along with the actual amount of CPU time used per kilobase in the scan of RFAMSEQ. (Note that the least accurate estimates are for filters with low fractions, which may be explained by the fact that we give up on statistical reliability of the filtering fraction when a reliable estimate would require too large a test sequence, e.g. greater than 10 Mbases.)

		Estimated	Actual
gcvT, model 1 (5 filters in series)	Filter 1 fraction	0.433	0.377
	Filter 2 fraction	0.229	0.194
	Filter 3 fraction	0.0824	0.0858
	Filter 4 fraction	0.00565	0.00727
	Filter 5 fraction	0.000314	0.000436
	Total run time (sec/Kb)	0.204	0.183
gcvT, model 2 (6 filters in series)	Filter 1 fraction	0.621	0.574
	Filter 2 fraction	0.3678	0.430
	Filter 3 fraction	0.0706	0.0913
	Filter 4 fraction	0.00254	0.00540
	Filter 5 fraction	0.000687	0.00139
	Filter 6 fraction	0.00005	0.000171
	Total run time (sec/Kb)	0.294	0.287
glmS (8 filters in series)	Filter 1 fraction	0.157	0.115
	Filter 2 fraction	0.0390	0.0266
	Filter 3 fraction	0.0183	0.0148
	Filter 4 fraction	0.00774	0.00447
	Filter 5 fraction	0.000858	0.00497
	Filter 6 fraction	0.000196	0.000117
	Filter 7 fraction	0.000015	0.0000125
	Filter 8 fraction	0	0.00000435
	Total run time (sec/Kb)	0.0310	0.0347

Chapter 6

RIGOROUS FILTERING FOR LOCAL CMS

In this chapter, I adapt the rigorous profile HMMs (of chapter 4) to local CMs. Local CMs have two advantages: (1) they can accommodate anomalous ncRNA structures, for example, tRNAs that are missing one arm of the classic tRNA cloverleaf structure, and (2) they allow the CM to match relatively conserved regions while ignoring highly diverged parts of the ncRNA homolog. Chapter 3 defined a simplified version of local CMs which are used in this chapter. This chapter may be difficult to read without reading section 4.2.

A component of a rigorous filter is an objective function to select filters that eliminate as much sequence as possible, as long as no homolog is discarded. My earlier objective function (the infinite-length forward algorithm) works poorly for local CMs. In this chapter, I create a new one that performs well, and has a clearer theoretical rationale. However, it is significantly slower than the infinite-length forward algorithm, and does not improve HMMs significantly in the non-local case.

I test the rigorous filters on local families in the Rfam Database.

6.1 Results

6.1.1 Rfam families

I scanned 9 local Rfam families using the local rigorous HMM. Filtering fractions were well below 0.01, except for RF00002, where the fraction is 0.0085. Six other families had fractions > 0.1, so could not be scanned. The remaining local families had more sequence conservation, so even BLAST should be fine.

6.1.2 Buried treasures

For 4 families, rigorous scans found putative homologs missed by the BLAST heuristic. There were 10 new hits in QUAD (RF00113), 1 in HgcC (RF00062), 10 in U15 (RF00067) and 88 in ribosomal S15 leader (RF00114). Little is known about HgcC, so it is difficult to evaluate the hit, although it does partially overlap a conserved hypothetical protein. Little is known about QUAD, but the 3 hits in *Salmonella* are plausible, given that this is a γ-proteobacterial family. Other hits in multicellular eukaryotes are likely false positives. Some of the lowest-scoring new U15 predictions overlap annotated repeat elements, so are likely wrong, but the higher scoring ones have little evidence for or against. Similarly, several new S15 leader (RF00114) hits contradict annotation, e.g., were upstream of the wrong type of gene. However, the top-scoring new hit is upstream of ribosomal protein S15, so is likely a true homolog.

6.2 Profile HMM filters

The mapping of CM to profile HMM is similar to the non-local case, as in chapter 4. The mapping itself is described in section 4.2. There are three developments to support the local case:

- The CM-to-HMM transformation is adapted to accommodate local features.

- The HMM Viterbi scanning algorithm is adapted to deal with local ends. In the CM's Viterbi algorithm, local ends are easy to support, but not in the case of HMMs.

- I present a novel method to select scores that filter as aggressively as possible, subject to the constraints. The new method performs much better on local CMs than the infinite-length forward algorithm, and has a clearer theoretical basis, although its optimization process is slower.

Table 6.1: CM parses correspond to HMM parses. The left column is rules used in a CM parse (from rat RNA of Figure 3.2). The right columns show the corresponding HMM parse, using HMM rules created from CM rules, by our transformation. (*) rules applying a local end to the rat RNA.

CM parse rules	**left HMM rules**	**right HMM rules**
$S_0 \rightarrow S_1$	$\overline{S}_0^L \rightarrow \overline{S}_1^L$	$\overline{S}_1^R \rightarrow \overline{S}_0^R$
$S_1 \rightarrow aS_2u$	$\overline{S}_1^L \rightarrow a\overline{S}_2^L$	$\overline{S}_2^R \rightarrow u\overline{S}_1^R$
$S_2 \rightarrow cS_3\epsilon$	$\overline{S}_2^L \rightarrow c\overline{S}_3^L$	$\overline{S}_3^R \rightarrow \epsilon\overline{S}_2^R$
(*) $S_3 \rightarrow cS_6$	$\overline{S}_3^L \rightarrow c\overline{S}_3^E$	$\overline{S}_3^E \rightarrow \overline{S}_3^R$
(*) $S_6 \rightarrow cS_6 \vert \cdots \vert uS_6$	$\overline{S}_3^E \rightarrow c\overline{S}_3^E \vert \cdots \vert u\overline{S}_3^E$	
(*) $S_6 \rightarrow \epsilon$	$\overline{S}_3^E \rightarrow \epsilon$	

6.2.1 From CM to profile HMM: think global, act local

The basic profile HMM grammar is the same as in the non-local case. For local begins, we create new HMM start and end states $\overline{S}_0^L$ and $\overline{S}_0^R$. $\overline{S}_0^L \rightarrow \overline{S}_1^L \vert \overline{S}_2^L \vert \cdots \vert \overline{S}_N^L$ and $\overline{S}_1^R, \overline{S}_2^R, \overline{S}_3^R \rightarrow \overline{S}_0^R$. (Note: the profile HMM cannot ensure that related left and right begins are used e.g., $\overline{S}_0^L \rightarrow \overline{S}_2^L$ can go with $\overline{S}_4^R \rightarrow \overline{S}_0^R$ in the same parse, which has no CM analog.)

For a local end at CM state S_i, we create HMM state $\overline{S}_i^E$ and rules $\overline{S}_i^L \rightarrow \overline{S}_i^E$ and $\overline{S}_i^E \rightarrow \{a, c, g, u\}\overline{S}_i^E \vert \overline{S}_i^R$. See Table 6.1 for an example.

6.2.2 Filtering with the profile HMM

As in the non-local case, we use the HMM to score each nucleotide position in the database sequence. If one of these scores exceed the threshold, the HMM has evidence that that position is the end of a homolog, so the CM algorithm is applied to a window (of size *window length*) ending at that nucleotide position.

Local ends present a challenge, since unbounded length sequences are allowed, and HMM algorithms have no window length parameter. Therefore, the HMM can match a part of a helix, then the rest of it billions of nucleotides later, thus inflating scores without any credible evidence of a homolog.

To limit this inflation, I limit HMM local ends to the CM's window length parameter. Naïvely, this would require adding an order of complexity to the algorithm, since we must store the scores in the last window-length positions for each $\overline{\mathrm{S}}_i^E$ state. (The CM does not have this problem; its Viterbi algorithm already deals with substring lengths explicitly.) However, using heaps [22] we can obtain log-time performance. (Briefly, the heap stores the previous W scores for each $\overline{\mathrm{S}}_i^E$ state; the heaps can efficiently find the highest score within the window.)

6.2.3 Faster filtering

Using heaps significantly speeds the use of local ends in the HMM Viterbi algorithm. Yet we can obtain a further overall ~30% speedup over heaps, by recognizing the following property. For this section, let the variable W be the CM's window length.

Dead scores. Suppose

- $\overline{\mathrm{S}}_i^E$ has Viterbi algorithm score s_1 at nucleotide k_1 and score s_2 at k_2,
- $k_2 > k_1$ and $s_2 \geq s_1$ and
- we are trying to score $\overline{\mathrm{S}}_i^R$ at some nucleotide k' to the right of k_1 and k_2, i.e., $k' \geq k_1$ and $k' \geq k_2$.

Since we want the highest score (for Viterbi), s_1 can be discarded immediately. To prove this, consider two cases. In the first case, both k_1 and k_2 are within W nucleotides of k', so we have to consider them both. Since $s_2 \geq s_1$, the maximum of the scores for $\overline{\mathrm{S}}_i^E$ must be at least equal to s_2, so it is safe to discard s_1. In the second case, k_1 is not within W nucleotides of k', in which case we clearly do not need it. Observe that there is no possibility that k_1 is within W of k' but not k_2, since $k_2 > k_1$. Whenever $k_2 > k_1$ and $s_2 \geq s_1$, we say that the score s_1 is *dead.*

Algorithm. The concept of dead scores allows a more efficient algorithm as follows. The algorithm uses two data structures: Q, a queue, and A, a sorted, circular array. (The reason that A is a circular array will become clear later; for now, imagine it is non-circular.) We maintain these data structures for each $\overline{\mathrm{S}}_i^E$ state in the HMM.

A stores live scores (i.e., scores not dead) and their nucleotide position. The data items in A are sorted to start with the highest score and end with the lowest. When a score for a new nucleotide position is added, binary search is used to determine its place in A. At this point, all entries in A after the new score (i.e., lower or equal scores), must be dead, so are removed from A. The new score is then inserted at this point into A.

The queue Q maintains pointers to each item in A. When a new score/position pair is added to the data structure, it is inserted into A as described, and entered into Q; its entry in Q points to its offset within A. When items in A are dead and deleted, their pointer in Q is set to null.

As the window of size W slides along, nucleotide positions will become dead because they are out of the window. To remove a position, it first leaves Q, and its pointer into A is inspected. If the pointer is null, the item is already dead, and nothing further needs be done.

If the pointer is non-null, we must remove the item from A. Conveniently, the item leaving A must be the highest score still live. Let the item leaving the queue have score s_1 at nucleotide k_1, and suppose to the contrary that there were some higher score still live in A: s_2 ($s_2 \geq s_1$) at nucleotide k_2. Since k_1 is about to leave the window, $k_2 > k_1$, and so s_1 must be dead already.

Since the leaving item has the highest score, it must be the entry in A with the lowest index. It is now that we require that A be a circular array. If A were a regular array, we could move all its entries one position left, which is needlessly inefficient. If A is a circular array, and keeps track of its first and last positions, then we need only increment its first position. (Of course, using a circular array makes the binary search slightly trickier.)

To summarize, we need to keep track of live scores for HMM state $\overline{\mathrm{S}}_i^E$ within a sliding window of length W. When a new score is calculated, we (1) perform a binary search in A to find its position, (2) mark each lesser-scoring item in A as invalid and adjust the "last"

pointer of circular array A, (3) insert the new score here and (4) enter the item into Q, storing a pointer to its copy in A. When a position goes out of the sliding window, it leaves Q. If its pointer into A is null, we are done. If the pointer not null, we increment the "first" pointer of A to remove it.

Run time. As noted previously, in practical terms this speeds scans by roughly 30%, based on Rfam families RF00002 and RF00003, on a 10-Kb test sequence. These family's window lengths are 180 and 250, which is moderate to large.

In asymptotic terms, let L be the biggest number of live items that the algorithm encounters at one time. Clearly, $L \leq W$. To scan a sequence of length N, maintaining scores for a single HMM state $\overline{\mathrm{S}}_i^E$, the algorithm takes time $O(N \log L)$:

- Each of the N nucleotides is entered and removed from structures A, Q exactly once. Except for the binary search, these operations take constant time, so in total these operations are linear in N.

- Each binary search is $O(\log L)$.

The expected-case run time is even better. In the expected-case analysis, I apply the simplifying assumption that the score at one nucleotide position is independent of other scores. Although this assumption is certainly not true, since adjacent positions score virtually the same subsequence, the following analysis will consider what is the highest score within a window—and simple correlations of scores seem unlikely to seriously compromise this analysis on average. In fact, since equal scores can be dead and adjacent positions often have equal scores, this analysis may overstate expected-case run time.

For expected case, we can use the average number of live items, $\overline{L}$, for expected complexity $O(N \log \overline{L})$. I show $\overline{L} = O(\log W)$.

Suppose we have W items in the window and ask how many are live. (It is unimportant how this changes as the window is shifted, just what is the expected number of live items at any instant.) Consider the W nucleotide positions $1, 2, \ldots, W$ and their scores $s_1, \ldots, s_W$. s_W is always live, since there is no position in the window greater than it. s_{W-1} is live if

and only if it is greater than s_W, so with probability $\frac{1}{2}$ (assuming independence of scores). Similarly, the probability that s_{W-2} is live is $\frac{1}{3}$, etc. By linearity of expectations, therefore, the expected number of live items is:

$$\begin{aligned}\overline{L} &= \sum_{i=1}^{W} \frac{1}{i} \\ &= O(\log W) \quad \text{sum of harmonic series [22, p. 44]}\end{aligned}$$

Thus, the expected run time of this algorithm is $O(N \log \overline{L}) = O(N \log \log W)$.

6.2.4 Constraints on scores for rigorousness

To ensure rigorous filtering, I previously defined constraints ensuring that the HMM's Viterbi score for any database subsequence upper bounds the CM's Viterbi score. (See chapter 4.) CM local begin/end rules also have HMM analogs, and result in one inequality each.

Suppose $S(\mathrm{S}_i \rightarrow \epsilon \mathrm{S}_N)$ is the logarithmic score of a local end rule in a CM. The related HMM rules are $\overline{\mathrm{S}}_i^L \rightarrow \epsilon \overline{\mathrm{S}}_i^E$ and $\overline{\mathrm{S}}_i^E \rightarrow \overline{\mathrm{S}}_i^R$. Let $S(\overline{\mathrm{S}}_i^L \rightarrow \epsilon \overline{\mathrm{S}}_i^E + \overline{\mathrm{S}}_i^E \rightarrow \overline{\mathrm{S}}_i^R)$ be the sum of their scores. (There is no need to distinguish them, since these HMM rules always go together in the Viterbi scanning algorithm.) Then, $S(\overline{\mathrm{S}}_i^L \rightarrow \epsilon \overline{\mathrm{S}}_i^E + \overline{\mathrm{S}}_i^E \rightarrow \overline{\mathrm{S}}_i^R) \geq S(\mathrm{S}_i \rightarrow \epsilon \mathrm{S}_N)$.

For local begins, consider $S(\mathrm{S}_0 \rightarrow \mathrm{S}_i)$, the score of a local begin CM rule. Then, $S(\overline{\mathrm{S}}_0^L \rightarrow \overline{\mathrm{S}}_i^L) + S(\overline{\mathrm{S}}_i^R \rightarrow \overline{\mathrm{S}}_0^R) \geq S(\mathrm{S}_0 \rightarrow \mathrm{S}_i)$. (In this case, there is a need to distinguish the rules, since the Viterbi scanning algorithm can use one rule without the other, since the profile HMMs do not enforce consistency.)

6.2.5 A new method to optimize rigorous HMMs

I consider a heuristic to improve filtering. Regardless of the heuristic to improve filtering, the inequality constraints must be satisfied, so rigorous filtering is guaranteed. In chapter 4, I developed the infinite-length forward algorithm objective function, but when adapted to the local case, the resulting profile HMMs do not filter well. I therefore developed a novel, and more general scheme. I also have an improved model for random sequences, that models variations in G+C content.

I now recap the general requirements for a rigorous filter objective function. To improve filtering, we need a method to estimate an HMM's likely filtering fraction. For example, we could measure the HMM's average filtering fraction on a random test sequence. This idea suffers two practical drawbacks. First, the expected fraction is not differentiable with respect to HMM scores, eliminating many optimization algorithms, e.g., gradient descent. Second, since filtering fractions can be low and represent relatively rare events, a fraction estimate can be unreliable, without using prohibitively large test sequences.

Minimizing expected Viterbi score

Instead of minimizing expected filtering fraction, I propose to minimize expected Viterbi score. If Viterbi scores are lower, more scores will be below the threshold, and therefore less sequence given to the CM. Also, expected Viterbi score can be estimated more reliably; each nucleotide contributes a number, whereas for filtering fraction, only a Boolean is contributed—usually false. Determining the Viterbi score is easy, but gradients are important to guide optimization.

First, given some test sequence (described below), we do backtraces at each nucleotide to get a set of Viterbi parses, $\pi_1, \pi_2, \ldots, \pi_n$, one for each nucleotide. The estimated expected Viterbi score is $E[V] = \frac{1}{n}\sum_{i=1}^{n} S(\pi_i)$, where $S(\pi_i)$ is a parse's score. A parse's score is the sum of scores of rules used in the parse. These rule scores correspond to variables, like $l_1, l_2, \ldots$ (used in chapter 4). So, $E[V] = \frac{1}{n}\sum k_j l_j$ for integers k_j. k_j is the number of times l_j's rule is used in $\pi_1, \ldots, \pi_n$. The k_j are easily counted from backtraces.

For derivatives, I assume $\frac{\mathrm{d}E[V]}{\mathrm{d}l_j} = k_j$. This is only an approximation, because as the l_j are changed, the Viterbi parses will eventually change, so the above equation for $E[V]$ also changes. (Thus, the objective function is not actually differentiable.) However, these pseudo-derivatives are a local approximation, and in practice guide solvers to minima.

Now we have a non-linear optimization problem: minimize $E[V]$ subject to the linear inequality constraints on the l_j. We solve this using CFSQP [77], although other non-linear program solvers could be used.

The test sequence is made as follows. %G+C can affect HMM filters (see section 5.4.2),

so I model %G+C variation in the test sequence. Given a target database (e.g., RFAMSEQ), we compute the G+C content of all 100-nucleotide blocks, yielding an empirical distribution. We then create a kind of stratified sample: in an n-nucleotide test sequence, the probability that the ith nucleotide is G or C is the i/nth quantile of the empirical distribution. I find $n = 50000$ leads to good performance, yet takes an acceptable amount of training time.

For the largest models (e.g., SRPs), training time can take a CPU week—still less than scan time on Gbase databases. However, training is still considerably slower than with the infinite-length forward algorithm.

In the case of the infinite-length forward algorithm, I proposed solving the non-linear program for each CM state's variables, holding other states' variables as fixed constants. This strategy is inappropriate with the expected Viterbi score objective function, since each evaluation requires scanning a moderately sized sequence. On the other hand, many CMs have too many variables to solve for at once, given current limits on computer RAM. So, my implementation collects variables of consecutive CM states until the total number of variables exceeds a constant (e.g., 800). Then the resulting non-linear program is solved, treating other variables as fixed, and a new set of variables is selected for optimization.

6.3 Discussion

This chapter extended rigorous profile HMMs to local CMs. The limited secondary structure techniques of the previous chapter could, in principle, be applied to local CMs. However, the sub-CM technique will require a larger window length in hairpins, which may degrade its speed.

This dissertation does not investigate the possibility. The ML-heuristic has proven itself highly sensitive on very challenging families, in non-local CMs (chapter 7). Although rigorous filters for local CMs are not powerful enough to test the ML-heuristic on harder local families, the heuristic's performance on moderately hard families suggest that it is good in these cases as well. Therefore, while more powerful rigorous filters will always offer advantages, I believe that the most-pressing need for local CMs is improved heuristics rather than improved rigorous filters.

Chapter 7

HEURISTIC PROFILE HMMS

This chapter describes a heuristic filter called the ML-heuristic ("Maximum-Likelihood heuristic"). As mentioned in the Introduction, the sensitivity and speed of the ML-heuristic is comparable to (though not as good as) highly tuned heuristics in the tRNAscan-SE program [82]. This suggests that generic heuristic filters are a reasonable solution to the problem of designing filters for the hundreds of families in Rfam, one that—unlike the creation of specialized tools like tRNAscan-SE—requires no human effort.

In the ML-heuristic, a profile HMM is created, in which HMM transition and emission probabilities are set to make the HMM maximally similar to the CM. (Chapter 5 added limited structure information to the profile HMM; heuristics should also benefit from analogous extensions, but I leave this to future work (See chapter 11).)

This chapter also introduces a novel methodology to evaluate heuristic filters. The perfect filter eliminates all sequences except the family members, which are preserved for the CM to find, but realistic heuristic filters must make a trade-off between missing true positives versus submitting too much to the CM (and therefore not accelerating searches). I analyze filters on real biological data, comparing the various heuristic methods to each other.

The results show that the ML-heuristic discriminates better than BLAST. Although BLAST itself runs faster than the profile HMM scan (~2-4 times faster), profile HMMs seem a better choice when high levels of sensitivity are demanded for challenging families, since overall scan time will be dominated by the CM. For a common example, to achieve maximal sensitivity for the cobalamin riboswitch (Rfam ID RF00174), BLAST requires the CM to scan 7% of RFAMSEQ, where our heuristics only require 0.001%, with overall scan time reduced by about 10 times. (The HMM itself is roughly 600 times faster than the CM.) In an extreme case, BLAST missed 90% of SECIS element homologs found with a

profile HMM filter in the same run time.

I also compare the ML-heuristic to tRNAscan-SE's heuristics. Despite the work that went into tRNAscan-SE—particularly the dedicated tRNA detectors used in its heuristics—the discriminative power of the ML-heuristic is similar to that of tRNAscan-SE's heuristics. This result is encouraging, since it suggests that the ML-heuristic could be used on other ncRNA families instead of designing family-specific heuristics. A weakness of the ML-heuristic is that it is a factor of 3-12 times slower than tRNAscan-SE's heuristics. However, its speed seems close enough to be immediately practical and further work can improve performance.

In summary, this chapter introduces the novel ML-heuristic, designed to make profile HMMs as close as possible to the CM. By contrast, previous algorithms in this dissertation set scores to facilitate rigorous filtering. I also consider filters in the context of heuristics, rather than the rigorous context of previous chapters; heuristics enable increased speed in exchange for slightly reduced sensitivity—an important practical advantage in many scenarios. This chapter presents a method to evaluate heuristics, and shows the ML-heuristic is the best available generic method, and is even comparable to tRNAscan-SE's specialized heuristics.

7.1 Results

I first compare profile HMMs to BLAST as a heuristic for CMs. Three types of profile HMMs are tested:

- ML-heuristic. The ML-heuristic was already mentioned, and is defined later in this chapter.

- Ignore-SS. Ignore-SS is defined in detail later in this chapter, but briefly it involves making a profile HMM directly from the input Multiple Sequence Alignment (MSA), ignoring any secondary structure in this input MSA. (By contrast, the ML-heuristic is designed to be as close as possible to the CM, and automatically captures any pseudocounts or other priors used in making the CM.)

- Rigorous profile HMMs (from chapter 4). Rigorous HMMs are used as heuristics by making their probability threshold a free parameter, instead of setting it to the CM's threshold.

For tRNAs, I also compare the ML-heuristic to tRNAscan-SE's heuristics.

7.1.1 ROC-like curves

I now compare profile HMMs and BLAST. Both filter types have a tunable parameter controlling selectivity. If BLAST's E-value cutoff is made more permissive, we can find more hits (according to the CM) at the expense of less selective filtering, and therefore more CPU time. Alternately, with a more restrictive E-value cutoff, the filter is more selectively but finds fewer hits. The profile HMMs have a score threshold parameter that can be used in a similar way to BLAST's E-value cutoff.

To visualize these trade-offs, I introduce the *ROC-like curve.* The ROC-like curve shows graphically how well the filter makes trade-offs between sensitivity and CPU time, and plots sensitivity versus filtering fraction. (Traditional ROC curves use 1 minus specificity in place of filtering fraction [90]; filtering fraction is an appropriate measure for sequence filtering since it measures the cost of a sensitivity level in terms of the amount of time that will be required by the CM.) I define filter sensitivity relative to the CM. Although the CM may predict ncRNAs that are not true homologs, it is roughly the best technology available, and the goal of filters is primarily to speed it; moreover, it is not practical to experimentally validate predictions.

By running profile HMMs on RFAMSEQ, we can plot sensitivity vs. filtering fraction at all possible score thresholds. (The algorithm will be described later, in section 7.3.) I applied a similar technique to BLAST. Unfortunately, it is difficult to know a priori what E-value threshold would cause the entire database to be scanned; instead, I used E-values that do not explore filtering fractions near 1. (Section 7.5.2 says more about ROC-like curves for BLAST.)

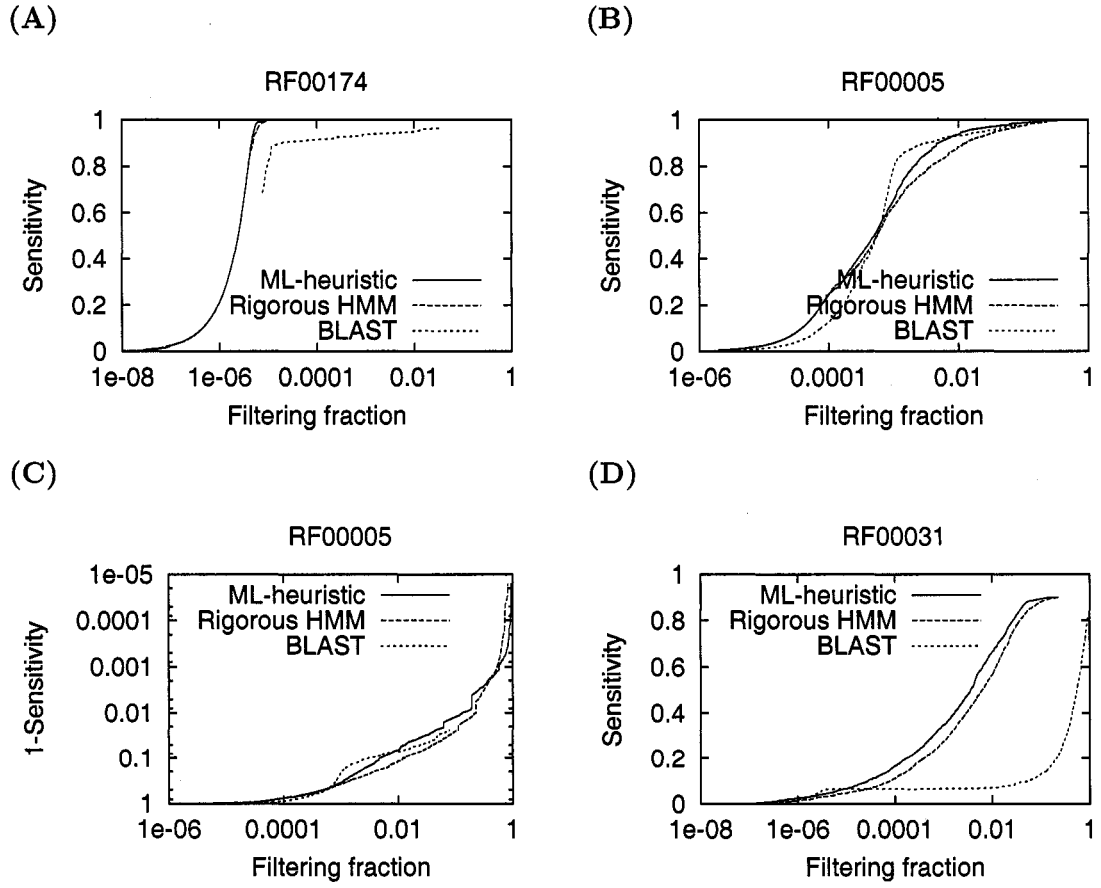


Figure 7.1: Selected ROC-like curves. All plot sensitivity against filtering fraction. Filtering fraction is in log scale, so the graphs do not resemble standard ROC curves. (A) This graph of RF00174 is typical of the other families; the ML-heuristic is slightly better than the rigorous profile HMM, and both often dramatically exceed BLAST. (B) Atypically, in RF00005, BLAST is superior, although only in one region. (C) This graph for RF00005 plots 1-sensitivity on log scale, for a better view of high levels of sensitivity; the rigorous profile HMM slightly beats the ML-heuristic at very high levels of sensitivity. (D) BLAST performs especially poorly for RF00031. (Recall that rigorous scans were not possible for RF00031, so only ~90% of hits are known; see text.) Section 7.5.6 includes all ROC-like curves, and the inferior ignore-SS heuristic.

7.1.2 Analysis of ROC-like curves

To test the heuristics on relatively difficult (highly structured) ncRNA families, I used families that could not be efficiently scanned using a rigorous profile HMM (i.e., the technique in chapter 4): 5S rRNA (Rfam ID RF00001), tRNA (RF00005), eubacterial RNase P (RF00010), the group II intron (RF00029), SECIS element (RF00031), and thiamin, lysine and cobalamin riboswitches (RF00059, RF00168, RF00174). Figure 7.1 shows a selection of the ROC-like curves. (The complete set is in section 7.5.6.)

Except for RF00031, I scanned these families using the sophisticated rigorous filters of chapter 5. Since rigorous filters were not feasible for RF00031, I scanned using all 4 heuristics (BLAST and HMMs) at relatively high filtering fractions, and treated the union of these scans' results as the complete list. Based on a raw CM scan of a subset of vertebrate sequences, I estimate that the heuristics' union contains ~90% of RF00031 hits.

In all cases, I found that ignore-SS was no better than the ML-heuristic, and was typically much worse. Although ignore-SS uses the input MSA in a similar way to the CM, pseudocounts affect the CM differently from the profile HMM, so ignore-SS only accurately reflects sequence information when there are many training examples. More generally, an advantage of the ML-heuristic is that it allows any scheme for transforming an input MSA into a probabilistic CM, since it only uses the CM. The ignore-SS vs. the ML-heuristic question is considered in more detail later, in section 7.5.4.

The rigorous profile HMM's performance was slightly better than the ML-heuristic in small parts of some ROC-like curves, but its performance was more often noticeably worse than the ML-heuristic. Since rigorous filters must guarantee perfect sensitivity, they optimize for rare RNAs. The ML-heuristic optimizes for average-case performance, so is expected to be usually more accurate.

Moreover, the process to create an ML-heuristic profile HMM takes about 1 second on a 2.8 GHz Pentium 4, while creating a rigorous profile HMM takes 30 seconds to several hours. Rigorous and ML-heuristic HMMs run at about the same speed; the ML-heuristic is preferred because it is faster to create, and its sensitivity (based on ROC-like curves) is more reliable. When rigorous HMMs are used rigorously, they guarantee perfect sensitivity, but

the ability to use a less stringent threshold gives heuristics a considerable speed advantage.

BLAST was almost always worse than the profile HMMs. In the most extreme case, the BLAST heuristic found only 10% of RF00031 hits even with a filtering fraction as high as 0.1, where the ML-heuristic found an estimated 90% of them at this filtering fraction. There was a small, but interesting part of the ROC curve for tRNA (RF00005) for which BLAST was better; the very large number of training tRNA sequences may benefit BLAST.

Running BLAST is typically 2-4 times faster than a profile HMM scan, so the BLAST heuristic may be preferred for especially easy families where a low filtering fraction still yields high sensitivity. However, the single best heuristic as measured by ROC-like curves seems to be the ML-heuristic. This is particularly relevant at higher levels of sensitivity on these difficult families, where the time spent running the CM would dominate overall scan time.

7.1.3 tRNAscan-SE heuristics

I compared ML-heuristic profile HMMs to tRNAscan-SE. tRNAscan-SE is used to predict tRNAs in most genome projects. Its heuristics are based on two previously created tRNA annotation programs that were selected based on superior sensitivity and selectivity, relative to other tRNA annotation programs (e.g., 7 programs are directly cited in the tRNAscan-SE paper [82]). Thus, tRNAscan-SE represents significant effort to create CM filters for a specific, well-studied family, and exploits a substantial body of work on tRNA detectors. I was interested to see how well the generic ML-heuristic compares to this specialized case.

Because tRNAscan-SE has many complex parameters, I avoid ROC-like curves, using only the default parameters. Three eukaryotic nuclear genomes were scanned (in all cases, organellar DNA was not used), as was archaeal and eubacterial DNA (all archaeal/eubacterial DNA in RFAMSEQ).

By default, tRNAscan-SE uses a window length of 500, but its heuristics can find intervals that are much smaller, and typically do. A window length of 500 is highly disadvantageous to the ML-heuristic (and other profile HMM techniques). For example, consider a 10-Kbase sequence with 5 widely separated tRNAs of length 100. With window length

Table 7.1: ML-heuristic vs. tRNAscan-SE heuristics. The first column names the genome sequences tested, followed by its size in megabases. The next column is the filtering fraction with tRNAscan-SE in its default settings (with domain of life specified on the command line); ML-heuristic scans were run at this same filtering fraction. The next 3 columns are sensitivity relative to rigorous scans for tRNAscan-SE ("t-SE"), the ML-heuristic with window length of 500 (W=500) and with window length 100 (W=100). (See text for why W=500 is worse.) The next 5 columns give measured run times, including the time used to run the CM. (Last column is without any filter. Second last is rigorous scan. By definition both CMs and rigorous filters have sensitivity 100% relative to the CM.) Most of the increase in profile HMM run time is because the profile HMM is slower than tRNAscan-SE's heuristics.

Sequence data	**Size (Mb)**	**Filtering fraction**	**Sensitivity of heuristic**			**Total run time, heuristic+CM (CPU hours)**				
			t-SE	**ML W=500**	**ML W=100**	**t-SE**	**ML W=500**	**ML W=100**	**rigorous scan**	**raw CM**
archaea	47	0.0034	98.5%	77.6%	99.3%	0.21	1.54	0.67	1.76	503.7
eubacteria	640	0.0034	99.4%	99.6%	99.8%	2.79	21.43	10.03	36.7	6553.2
C. elegans	100	0.0012	98.1%	55.1%	97.5%	0.13	3.42	1.03	64.3	1056.1
Drosophila	117	0.00036	99.7%	56.9%	99.3%	0.08	1.33	1.12	19.0	1233.3
human	3070	0.00055	83.4%	86.8%	90.4%	3.41	53.75	30.86	581.1	32422.6

100, and filtering fraction 500/10000, a profile HMM could potentially select all 5 tRNAs, attaining 100% sensitivity. However, with that same filtering fraction, but a window length 500, even a perfect HMM could attain no better than 20% sensitivity, since only one of the tRNA hits (with 400 extraneous flanking nucleotides) could be reported without exceeding the filtering limit. This explains the reduced sensitivity of the W=500 case in Table 7.1. So, I also tried a window length of 100, the value used for Rfam's tRNA family. In all cases, the CM's window length was 500. (Overlapping windows of length 100 can create larger interval sizes.)

The filtering fraction of tRNAscan-SE was measured on each of the test genomic databases, ML-heuristic profile HMMs were run at the same filtering fraction, and run time and sensitivity were measured; see Table 7.1.

This sensitivity measurement is the heuristic sensitivity relative to the CM, not the sensitivity relative to any experimental criteria. Both heuristics have low sensitivity on human, and tRNAscan-SE is lower. A concern is that the ML-heuristic has simply found more false positives. Although it is not practical to experimentally characterize all predictions, tRNAscan-SE has additional heuristics to predict pseudo-tRNAs. A practical strategy is to use these pseudo-tRNA predictions as a rough proxy for false positives.

The HMM finds 64 hits that tRNAscan-SE does not. tRNAscan-SE's pseudo-tRNAs heuristics indicate that 29/64 are likely pseudo-tRNAs. Of the 16 putative tRNAs found with tRNAscan-SE but not HMMs, 16/16 are predicted pseudo-tRNAs. Among the 592 hits common to tRNAscan-SE and the HMM, 91 are likely pseudo-tRNAs, so the change in number of pseudo-tRNAs seems modest.

For example, assuming that the pseudo-tRNA predictions are equivalent to the number of false positives, we can measure the filters' positive predictive value (PPV), i.e., the number of true positive predictions divided by the total number of positive predictions (true and false). Then the ML-heuristic's PPV is $1-(29+91)/(64+592) \approx 81.7\%$, versus tRNAscan-SE's PPV of $1-(16+91)/(16+592) \approx 82.4\%$. Thus, unless tRNAscan-SE's pseudo-tRNA heuristics are radically different from false positives, it would appear that tRNAscan-SE's PPV is better, but only marginally.

In conclusion, tRNAscan-SE's heuristics seem preferable to the ML-heuristic for tRNA

detection, certainly in speed. However, if the results extrapolate to other ncRNA families, they suggest that generic heuristics are a more cost-effective solution for these other families than family-specific tools. The sensitivity and positive predictive value of the ML-heuristic is similar to that of tRNAscan-SE, and the speed is in the same league, and still practical. Since most other ncRNA families have no detection tools like those in tRNAscan-SE's heuristics, the ML-heuristic would require significantly less work than a family-specific scheme, yet can be expected to provide comparable results.

7.2 The ML-heuristic profile HMM

In this section, for the purposes of exposition, I use the probabilistic form of CMs (not odds ratios or logarithmic scores).

7.2.1 Profile HMM grammar

The profile HMM grammar is created from the CM in exactly the same way as for rigorous profile HMMs, as in chapter 4. However, previously we assigned scores in order to ensure rigorous filtering. I now discard rigorousness in favor of a heuristic.

Ignore-SS. The CM-to-HMM grammar transformation is equivalent to a transformation of MSAs. Given the MSA (with secondary structure) used to create a CM, a profile HMM can be created by removing the secondary structure (i.e., base pair annotations), and using the CM creation method (section 3.3). With no structure, this "CM" will be equivalent to a profile HMM; the profile HMM will capture the primary sequence information of the original MSA, but not its secondary structure. I call this strategy the Ignore-SS heuristic.

ML-heuristic. In the ML-heuristic, we assign probabilities to the profile HMM so that it is as similar as possible to the CM, to make the heuristic more accurate. The ignore-SS heuristic trains a profile HMM on the MSA used to create the CM. Unfortunately, issues like pseudocounts affect the profile HMM differently from the CM, so the resulting profile HMM is not as similar as it could be. To avoid this problem with pseudocounts (or other priors), I propose training the HMM directly from the CM. (Results show empirically that

training from the CM is more accurate than from the MSA; the issue is further treated in section 7.5.4.)

Suppose we generate a random MSA from the CM. If this MSA has many sequences, it accurately reflects the CM's probability distribution. We can now learn a profile HMM from this MSA without pseudocounting, i.e., using maximum likelihood. The larger the MSA, the more closely the profile HMM's distribution is to that of the CM, at least in the positional sequence information that the profile HMM can model. In fact, it is possible to simulate an MSA with infinitely many sequences, i.e., the limiting case.

The correspondence between CM and profile HMM in the MSA is the same as the correspondence between rules. Suppose a sequence in the MSA uses CM rule $\mathrm{S}_i \rightarrow x_L \mathrm{S}_{i+1} x_R$. With the MSA structure removed, this will correspond to profile HMM rules $\overline{\mathrm{S}}_i^L \rightarrow x_L \overline{\mathrm{S}}_{i+1}^L$ and $\overline{\mathrm{S}}_i^R \rightarrow x_R \overline{\mathrm{S}}_{i-1}^R$.

The counts used to set the probability for $\overline{\mathrm{S}}_i^L \rightarrow x_L \overline{\mathrm{S}}_{i+1}^L$ should be proportional to the frequency with which this rule is used by the profile HMM in parsing sequences from the CM-generated infinite MSA. This HMM rule is used for CM rules $\mathrm{S}_i \rightarrow x_L \mathrm{S}_{i+1} x_R$ for any x_R, which are emitted in the MSA with frequency $\Pr(\mathrm{S}_i \rightarrow x_L \mathrm{S}_{i+1} x_R)$. Therefore, the rule counts are

$$\mathcal{C}(\overline{\mathrm{S}}_i^L \rightarrow x_L \overline{\mathrm{S}}_{i+1}^L) \propto \sum_{x_R \in \{a,c,g,u,\epsilon\}} \Pr(\mathrm{S}_i \rightarrow x_L \mathrm{S}_{i+1} x_R)$$

The virtual counts are then normalized into probabilities:

$$\Pr(\overline{\mathrm{S}}_i^L \rightarrow x_L \overline{\mathrm{S}}_{i+1}^L) = \frac{\mathcal{C}(\overline{\mathrm{S}}_i^L \rightarrow x_L \overline{\mathrm{S}}_{i+1}^L)}{\sum_{x \in \{a,c,g,u,\epsilon\}} \mathcal{C}(\overline{\mathrm{S}}_i^L \rightarrow x \overline{\mathrm{S}}_{i+1}^L)}$$

Similar equations are used for the right-side HMM rule. (The algorithm is somewhat more complicated with fully general CMs; see section 7.5.5.) This method of setting profile HMM probabilities can be viewed as learning a maximum likelihood profile HMM from the distribution of MSAs induced by the CM.

7.2.2 Filtering with the profile HMM

The actual filtering algorithm is as in section 4.2 (and technical details in section 4.4.3). However, (for exposition only) this chapter uses a probability threshold (since probabilities

are more convenient for the main exposition). The HMM's threshold is totally independent of the CM's threshold.

7.3 Calculating ROC-like curves

In Results, I compared profile HMMs and BLAST using ROC-like curves, which plots sensitivity vs. filtering fraction at every threshold.

It is possible to calculate a ROC-like curve via a single scan of the database. Suppose a heuristic probability threshold is chosen, and consider under what circumstances a given nucleotide position will be scanned by the CM (i.e., will be in the numerator of the filtering fraction): the position will be scanned if there is some position up to W (window length) nucleotides to the right whose probability is above the heuristic threshold. Equivalently, the position will be scanned if the maximum of the probabilities within the W nucleotides to the right is above the threshold. So, for each nucleotide position, we compute the maximum of the probabilities of its W right neighbors, calling this the position's *inclusion point.* To obtain a filtering fraction f, we select a threshold that is less than a fraction f of the inclusion points. Thus, keeping a sorted list of inclusion points allows us to quickly look up a threshold for a given fraction, or the reverse.

In Results, I also analyzed the sensitivity of heuristics relative to ncRNAs detected by a given CM: how many of these can the heuristic filter detect at a given heuristic threshold? A given ncRNA will be detected by the filter—and submitted to the CM—if the inclusion points of each nucleotide within the ncRNA are above the selected heuristic threshold. Thus, for each ncRNA, we can calculate the heuristic threshold necessary to detect it. To find ncRNAs detectable by the CM, I use a rigorous scan.

I used this scheme to plot the filtering fraction and sensitivity of heuristics at different heuristic thresholds. The scheme can also be used for the BLAST heuristic if we define analogous inclusion E-values for BLAST. (However, some technical complications with this are addressed in section 7.5.2.)

7.4 Discussion

Profile HMMs appear to have superior accuracy to BLAST, although the region in Rfam family RF00005 where BLAST was superior suggests possible advantages with many family members. BLAST itself runs 2-4 faster than the HMM, making it a logical tool for families with high sequence conservation where BLAST's accuracy is ample. Tuning of BLAST, e.g., tuning the DNA substitution matrix, may improve its overall accuracy. In using BLAST, there are two main heuristics being used: BLAST's word-matching heuristic to seed gapped alignments, and the gapped alignments used as a heuristic for CMs. It is unclear which heuristic is hurting sensitivity. The issue is subtle because, even though most database subsequences have an exact word match to at least some known family member, it may not match the most useful member for the alignment phase.

It would be interesting to test a pure Smith-Waterman heuristic. However, this will be very slow, since Smith-Waterman will be performed on each of the known family members—typically tens or hundreds of RNAs.

I was surprised by how competitive ML-heuristic profile HMMs are to tRNAscan-SE's heuristics, given that tRNAscan-SE was explicitly designed to detect tRNAs, whereas the ML-heuristic applies equally to any ncRNA family. The profile HMM itself is slower than tRNAscan-SE's heuristics, though not in a totally different class. For eukaryotes, where tRNAscan-SE has a low filtering fraction, tRNAscan-SE is overall much faster; for prokaryotes, where the CM is run more often, the difference is less. In terms of sensitivity at a given filtering fraction, the heuristics were comparable, usually within a fraction of a percentage point of each other.

These results suggest that the ML-heuristic is generally a preferred method for heuristic filters of new ncRNA families for which creation of a family-specific filter would be a large amount of work. Although some families may have features different from tRNAs that family-specific filters could exploit, it is likely that such features could be integrated into a generic filter. Moreover, there is clearly room for improvement to the ML-heuristic, such as augmenting the profile HMM with structural information, as in chapter 5.

My heuristics make scans for SECIS elements (RF00031) 10 times more sensitive than

when the BLAST heuristic is used. This allowed the detection of known, more diverged SECIS elements. Although I did not carefully evaluate the additional ~2200 new hits, many are upstream of known selenoproteins, and therefore probably true positives. One of these is the first known viral SECIS [114]. Some of these hits are likely false positives, e.g., the 165 prokaryotic hits. Overall, this may represent an improved SECIS element finder that could be used for a selenoprotein detection pipeline [67]. This possibility is evaluated further in chapter 11.

The ML-heuristic is extended to local CMs in chapter 8.

In summary, in this chapter I designed a new heuristic filter for CM searches, the ML-heuristic, and showed it superior to previous heuristics. Moreover, results indicate a rational basis for selecting its score threshold; even for difficult families, a filtering fraction of 0.01 was sufficient to find the majority of family members and would provide a roughly hundred-fold speedup. I have also shown a method to evaluate a heuristic CM filter, using a ROC-like curve, and how results of rigorous scans can be used to measure the sensitivity of the filter itself.

7.5 Additional details

This section provides supplementary technical details and results, specifically:

- How to set heuristic thresholds.
- How to calculate ROC-like curves for BLAST.
- How to calculate ROC-like curves for profile HMMs for very large databases like RFAMSEQ.
- Why it is important to train from the CM and not the MSA.
- How to extend the ML-heuristic technique to fully general CMs, as opposed to the simplified model considered in the main text.
- Complete set of ROC-like curves for all 8 families tested.

7.5.1 How to set heuristic thresholds

As noted earlier, a reasonable approach to selecting a heuristic threshold is to find a threshold yielding a small filtering fraction like 0.01. In general, when the filtering fraction gets very small, the profile HMM scan time will start to dominate the overall scan time; therefore there is little incentive to reduce the filtering fraction further. On the other hand, high filtering fractions require that the CM is run too often, making scan time prohibitive. Therefore, a fixed filtering fraction is a rational choice. (There are many other reasonable strategies, e.g., using a filtering fraction that allows the HMM to find all the seed members.)

To calculate a heuristic threshold yielding a desired filtering fraction, we could use the same technique for generating ROC curves: the simplest approach would be to run the profile HMM on the entire database to collect inclusion points, then find the appropriate heuristic threshold, and re-scan. Even if the profile HMM scan takes half of the overall scan time, this would only increase total scan time by 50%. However, it does lead to very large files when run on large genome databases given the current techniques for calculating ROC curves.

An alternative is to scan some representative sequences, and estimate based on that. I have observed that database G+C content tends to account for much of the variation in filtering fraction, as in section 5.4.2. Therefore, I use N test sequences, assuming the database contains a given proportion of each. For RFAMSEQ, I use $N = 3$ test sequences: a 1:1:1 ratio of *Bordetella*, *E. coli* and *Staphylococcus* genomes. For a prokaryotic database, a 1:2:4 ratio is better.

Each genome is scanned, storing inclusion points. Then the computer iterates through heuristic thresholds starting at the highest (most stringent). At each proposed heuristic threshold, the inclusion points allow a quick calculation of the filtering fraction for each of the three test genomes. Based on the assumption about the composition of the database, it is easy to estimate the filtering fraction on the overall database, extrapolating from the test genomes. The iteration over heuristic thresholds stops when the threshold is found that is estimated to yield the desired filtering fraction on the genome database. (A binary search strategy would reduce the time taken by this step. However, the process of collecting

inclusion points is linear in the number of nucleotides scanned, so a binary search strategy cannot reduce overall CPU time significantly.)

7.5.2 ROC-like curves for BLAST

In computing ROC-like curves for BLAST, I assumed that any hit with E-value e will be found if and only if the E-value cutoff is set at e' for any $e' > e$. In fact, this is not strictly true.

BLAST uses exact word matches to seed searches, but also uses ungapped alignments to seed the more expensive gapped alignments. These ungapped alignments are treated at the same E-value cutoff, which may violate my assumption. For example, suppose we run BLAST at E-value cutoff 1000, and find a hit with E-value 5. If we run BLAST with E-value cutoff 10, we expect to again obtain this hit. However, the hit's E-value in the ungapped alignment phase may be greater than 10, and therefore the hit may not be discovered.

Clearly, this is unlikely to be a significant factor. In fact, the assumption may even lead to an overstatement of BLAST's capabilities, since it effectively improves the sensitivity of BLAST ungapped alignment phase.

To be sure, I ran BLAST with lower E-value cutoffs and found that predicted sensitivity and filtering fractions differed from the actual values by an insignificant amount. For example, for RF00005 the graphs shown in this chapter were generated with an E-value cutoff of 1000. I re-ran RF00005 with a cutoff E-value of 1. The ROC-like curves for these two invocations of BLAST overlapped completely to the available resolution. Thus, empirically, the error introduced by the incorrect assumption about BLAST's E-values is not significant.

7.5.3 Calculating ROC-like curves with very large databases

The curves in this chapter were generated by scanning the roughly 8-Gbase RFAMSEQ database from the Rfam database version 5.0. This yields about $1.6 \cdot 10^{10}$ inclusion points (since each nucleotide is scanned twice, once for each DNA strand). To reduce computer memory demands, I stored these in an associative array mapping an inclusion point (as a C/C++ 'float' data type) to an integer representing the number of times that inclusion point

was observed. The associative array implementation used the C++ Standard Template Library 'map' class, which is based on red-black trees. Using this simple scheme, the memory requirements were still large—in some cases, I observed the process using over 1 GB of RAM—but were practical for these tests.

If necessary, the memory requirements could be reduced further by storing the inclusion points with less precision. This precision is probably not necessary for either accurate ROC-like curves, or for reasonable estimates of score thresholds for a given filtering fraction. Other data structures may also reduce memory requirements.

7.5.4 Training profile HMMs from the CM versus from the MSA

A simple method to transform an MSA into an HMM is to erase the structure annotation, and run the standard algorithm to learn a profile HMM [27], which is the Ignore-SS heuristic. Unfortunately, when pseudocounts (or other priors) are used, the resulting profile HMM does not capture the same primary sequence information as the CM does. This phenomenon arises since the pseudocounts will affect the profile HMM differently from the CM. (This explains the poor performance of the ignore-SS heuristic for families that do not contain enough sequences to overwhelm the pseudocounts.)

Example. I illustrate the issue with an example. Suppose we have an MSA with two columns, and these columns are base paired. For extra simplicity, I ignore insertions and deletions, and consider only the two nucleotides A and C (so I am assuming a two-letter alphabet). The MSA contains one sequence: an A-A pair. Finally, suppose we apply the simple plus-one pseudocount, i.e., Laplace's rule [27].

Table 7.2 summarizes the various calculations. In the CM, with plus-one pseudocounts, A-A will have probability $\frac{2}{5}$, while other base pairs will have probability $\frac{1}{5}$.

Now we apply the ignore-SS heuristic: we create profile HMM from the MSA, using the same plus-one pseudocount. In the first column (the left nucleotide of the base pair), the probability of A is $\frac{2}{3}$, and the probability of C is $\frac{1}{3}$.

Applying the ML-heuristic to the CM, the probability of A is $\frac{3}{5}$, and the probability of C is $\frac{2}{5}$. These probabilities are clearly different from those calculated under ignore-SS.

Table 7.2: Training heuristic HMMs: from CM or from MSA. See text for explanation. (A) straight counts of the sequences—one A-A pair. (B) CM probabilities with plus-one pseudocounts. (C) Probabilities of the left HMM position, using the ignore-SS heuristic and with plus-one pseudocounts (trained from the MSA). (D) Probabilities of the left HMM position, using the ML-heuristic (trained from the CM).

(A)

	A	C
A	1	0
C	0	0

(B)

	A	C
A	$\frac{2}{5}$	$\frac{1}{5}$
C	$\frac{1}{5}$	$\frac{1}{5}$

(C)

A	$\frac{2}{3}$
C	$\frac{1}{3}$

(D)

A	$\frac{3}{5}$
C	$\frac{2}{5}$

The reason is that, since there are more events in the CM, the effective weight of the pseudocount is higher for CMs than for HMMs. The difference between $\frac{2}{3}$ (≈ 0.67) and $\frac{3}{5}$ ($= 0.6$) is modest, but can significantly affect accuracy. Moreover, if we had used all four nucleotides, with the same single sequence MSA of A-A, the probabilities would be $\frac{2}{5}$ ($= 0.4$) and $\frac{5}{17}$ (≈ 0.29).

Thus, when pseudocounts are used, ignore-SS yields different probabilities from the ML-heuristic. Although this example demonstrates that the two heuristics yield different results, the example does not indicate which is better.

Which is better? I conclude that the ML-heuristic is better, based primarily on empirical results. Empirically, our ROC-like curves show that the ML-heuristic is better than ignore-SS. Moreover, the cases in which their accuracy is comparable occur when there are many known family members, so the pseudocounts are irrelevant, and the two heuristics result in approximately the same probabilities.

From a theoretical perspective, there are reasons to believe that it is best for the profile HMM to follow the CM as closely as possible, and reasons to believe it should be independent (i.e., use priors that are ignorant of the CM's use of priors). I discuss each of these positions, and then analyze them.

For the argument in favor of ignore-SS, consider an extreme case: an oracle filter that perfectly eliminates everything but the ncRNA homologs. Such a filter is the ideal case, and will yield perfect sensitivity and specificity, even though it completely ignores the CM.

Although this ideal filter is not attainable, we may claim that the ignore-SS heuristic, using profile HMM-appropriate priors, is better at finding ncRNAs, and therefore closer to the ideal filter. This argument, of course, requires that the profile HMM's priors are truly better for finding ncRNA homologs, which is doubtful.

However, there are reasons (in addition to empirical results) to expect that following the CM is better (i.e., that the ML-heuristic should prove superior). First, ncRNAs detected by the profile HMM that are not detected by the CM are essentially useless; especially since we are evaluating sensitivity relative to the CM, it is advantageous to bias the profile HMM to the CM.

Second, the CM can encode information about ncRNAs that is not accessible to a standard profile HMM. This will be especially relevant for CMs using sophisticated, RNA-aware priors. For example, such a CM would reflect different mutation rates for nucleotides that are base paired versus unpaired nucleotides; in some cases, this distinction may be encoded in a prior. If a profile HMM ignores the priors, it will be losing this information. The ML-heuristic, by basing its probabilities directly on the CM, exploits any priors used in the CM. Better priors trained on RNA alignments are already used in the RSEARCH program [61], and I anticipate further developments in this area.

In summary, there are theoretical arguments to support either ignore-SS or ML-heuristic. I find the ML-heuristics more compelling because it is unclear whether the ideal filter proposed in the ignore-SS argument will be relevant in practice. Moreover, the ML-heuristic argument will be more significant as CM priors are improved. Overall, the empirical evidence supporting the ML-heuristic is the most persuasive evidence.

7.5.5 ML-heuristic with fully general CMs

Handling general CM grammars

The ML-heuristic uses precisely the same profile HMM grammar as the rigorous profile HMM filters of chapter 4. In those terms, the ML-heuristic uses the *compact-type* HMM. (The alternative is the *expanded-type* HMM, which uses additional states. These additional states slow scan speed slightly, but for rigorous filter often reduce the filtering fraction very

significantly. However, for the ML-heuristic, expanded-type HMMs appeared to offer no accuracy benefit, as demonstrated in Figures 7.4 and 7.5, below.)

The computation of the HMM probabilities for the ML-heuristic is somewhat more complicated in the case of fully general CMs. With general CMs, the profile HMM grammar includes insert states, which have self loops. Also, the $\overline{\mathrm{S}}_i^L$ and $\overline{\mathrm{S}}_i^R$ states are split into separate states based on whether they are emitting (emitting a, c, g or u) or a non-emitting deletion state (emitting ϵ). The CM states S_i are split into 2 or as many as 4 states in a similar manner.

A consequence of these additions is that a given CM state need not be entered when it generates a sequence in the infinite MSA. Also, some CM states—namely the insert states—may be entered more than once when generating a sequence. To account for this, we calculate the expected number of times that each CM state will be entered in generating one sequence.

Another consequence is that CM parses become more complicated. General CMs are conceptually divided into *nodes*. A node relates to the S_i used earlier in this chapter, but each node has several states, including one or two insert states. I therefore consider *sub-parses*, which start at one CM node and end at the next. Each sub-parse consists of one or more CM rules. The CM rules can be mapped to the corresponding profile HMM rules, i.e., the profile HMM rules that would be used if the infinite MSA is parsed without structure.

I now define the method to set profile HMM probabilities for the ML-heuristic in the general case. Let $\overline{\mathrm{S}} \rightarrow x\overline{\mathrm{S}'}$ be any profile HMM rule. Let $\pi(\overline{\mathrm{S}} \rightarrow x\overline{\mathrm{S}'})$ be the set of CM sub-parses that would use profile HMM rule $\overline{\mathrm{S}} \rightarrow x\overline{\mathrm{S}'}$. For any one sub-parse π, let π_1 be the first state used in the parse. Let $E(\pi_1)$ be the expected number of times the first state of sub-parse π is entered, and $\Pr(\pi)$ be the probability of the parse assuming this first state is entered, i.e., simply the product of the probabilities of the rules in π.

Then, the count for profile HMM rule $\overline{\mathrm{S}} \rightarrow x\overline{\mathrm{S}'}$ is:

$$C(\overline{\mathrm{S}} \rightarrow x\overline{\mathrm{S}'}) = \sum_{\pi \in \pi(\overline{\mathrm{S}} \rightarrow x\overline{\mathrm{S}'})} E(\pi_1) Pr(\pi)$$

This equation yields the expected number of times the profile HMM rule will be used to parse a sequence from the infinite MSA without structure. I note that the set $\pi(\overline{\mathrm{S}} \rightarrow x\overline{\mathrm{S}'})$

is infinite in size because it can include an unbounded number of insert state self loops. Fortunately, the contribution of the unbounded series of insert states is easily summed, since it is simply the sum of an infinite geometric series.

The equation does not, however, set profile HMM insert state self loop rules correctly, since profile HMM insert states can be used multiple times within one sequence. So, for insert states, the following rule is used:

$$\mathcal{C}(\overline{\mathrm{S}} \rightarrow x\overline{\mathrm{S}}) = E(\mathrm{S}) \Pr(\mathrm{S} \rightarrow x\mathrm{S})$$

where S is the CM state corresponding to HMM insert state $\overline{\mathrm{S}}$.

Once these virtual counts $\mathcal{C}$ are computed, they are normalized into probabilities as before.

Odds ratios vs. probabilities

I described the ML-heuristic under the assumption that CMs are straight probabilistic models. However, in fact their emission scores are set from odds ratios. Typically, the numerator in the odds ratio is the CM-assigned probability, and the denominator (null model) is a 0th-order Markov model. My technique generalizes to this case easily: we normalize the CM probabilities to extract the straight-CM probabilities (the numerator), then create an ML-heuristic profile HMM. Finally, we apply the null model (the denominator) to the profile HMM, to convert it to odds ratios.

7.5.6 Complete set of ROC-like curves

There are 16 figures, two for each family discussed earlier in this chapter. One figure uses log scale for plotting 1-sensitivity, to show extra detail at high levels of sensitivity. The other does not. Both of these graph types were shown earlier (Figure 7.1).

The curve labels are as follows. "ML path" is the ML-heuristic profile HMM. "Rigorous HMM" is the expanded-type rigorous profile HMM. "ignore-SS" is the ignore-SS profile HMM.

For BLAST, the set of known ncRNAs used is either "SEED" or "90%". "SEED" means the list of known ncRNAs was used on which the CM was trained—this is the normal case.

"90%" means that the full set of ncRNAs found with BLAST+CM on the Rfam site was used, but sequences were discarded that were more than 90% similar. (This corresponds to the sequences in Rfam.fasta.gz, distributed by Rfam [48]. This gives BLAST an unfair advantage, but it's interesting to see the results.) "1-sided" means that whenever BLAST found a hit, only the hit's DNA strand would be scanned by the CM. (This is the scheme used previously in this chapter.) With "2-sided", both strands of a BLAST hit are scanned. (This is the current technique in the Rfam database.)

"ML-path (full 90% id)" is the ML-heuristic trained on the 90% of the full ncRNA family—the same set as for "90%" in the BLAST curves.

Some scans were not done; in this case, the curves are missing from the graphs.

The graphs support a number of conclusions:

- The expanded-type ML-heuristic does not discriminate better than the compact-type. I compared them for Rfam's tRNA family (RF00005; Figures 7.4 and 7.5, "ML path" and "expanded ML path"). Since the lines appear to overlap, there was no significant difference in performance on ROC-like curves for the two profile HMM types.

- The ignore-SS heuristic is inferior to the other profile HMMs when few seed members are available. For example for RF00010 (Figures 7.6 and 7.7), ignore-SS requires filtering fractions that are about 10^4 times higher than those of the other techniques for the same level of sensitivity. When more training examples are available, ignore-SS produces comparable results, e.g. RF00005 (Figures 7.4 and 7.5).

- The ML heuristic outperforms rigorous profile HMMs in most cases. In some cases, the two heuristics' curves approximately overlap. For example, for RF00001 (Figure 7.2) they overlap almost entirely, although Figure 7.3 shows that at the highest levels of sensitivity, the ML-heuristic is better. In many families, the ML-heuristic is generally better, particularly RF00031 (Figures 7.10 and 7.11; note that the rigorous filter is dark blue in these figures).

  For the tRNA family, their relative performance varies: for most of the ROC-curve the ML-heuristic outperforms rigorous filters (Figures 7.4 and 7.5), but at very high

sensitivity and filtering fractions close to 1, the rigorous filters are superior (Figure 7.5). Overall, the ML-heuristic appears better in terms of sensitivity measured by ROC-like curves.

- 1-sided BLAST searching is better in discriminative power than 2-sided BLAST. In every ROC-like curve, the 1-sided heuristic is superior at every point to the 2-sided heuristic, often by a factor close to 2.

  This is intuitive, since when BLAST detects an approximate match on one strand, there is no biological reason to necessarily expect a hit on the opposite strand, unless BLAST also finds an approximate match on that strand. Therefore, up to half of the sequence returned by a 2-sided BLAST filter has little evidence for an ncRNA hit. I note that it is easier to implement a 2-sided BLAST heuristic, because a sequence can directly be submitted to a CM scanning program.

- With both profile HMMs and BLAST, I used either the seed members or the "full 90% id" set. The use of the "full 90% id" set yields an unrealistic advantage, since in practice only the seed members would be known.

  I tested the "full 90% id" for profile HMMs only for Rfam's tRNA family (Figures 7.4 and 7.5). At the relatively high levels of sensitivity that are of interest, the "full 90% id" profile HMM does better than the profile HMM trained only on the seed members.

  BLAST took more advantage of the "full 90% id" set, in that the difference between its performance with either set was larger. A particular difference was for Rfam's tRNA family (Figures 7.4 and 7.5), where BLAST was able to take dramatic advantage of the "full 90% id" set, particularly at high levels of sensitivity.

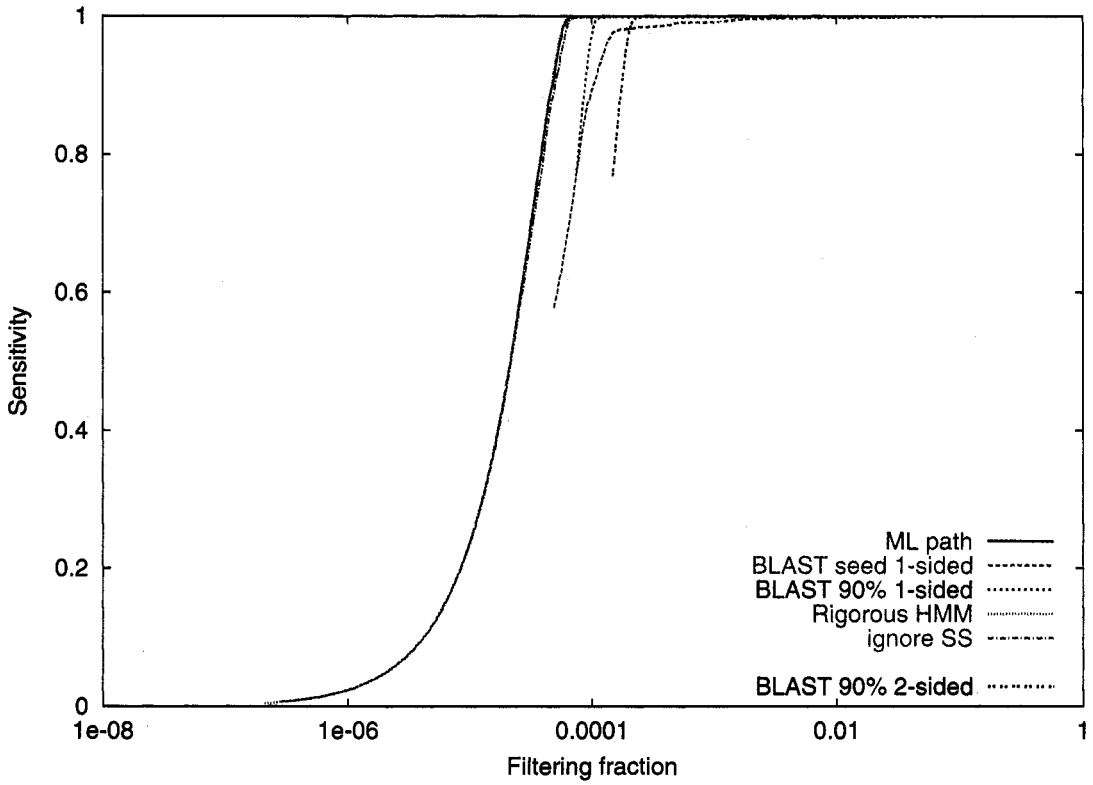


Figure 7.2: ROC-like curve: RF00001

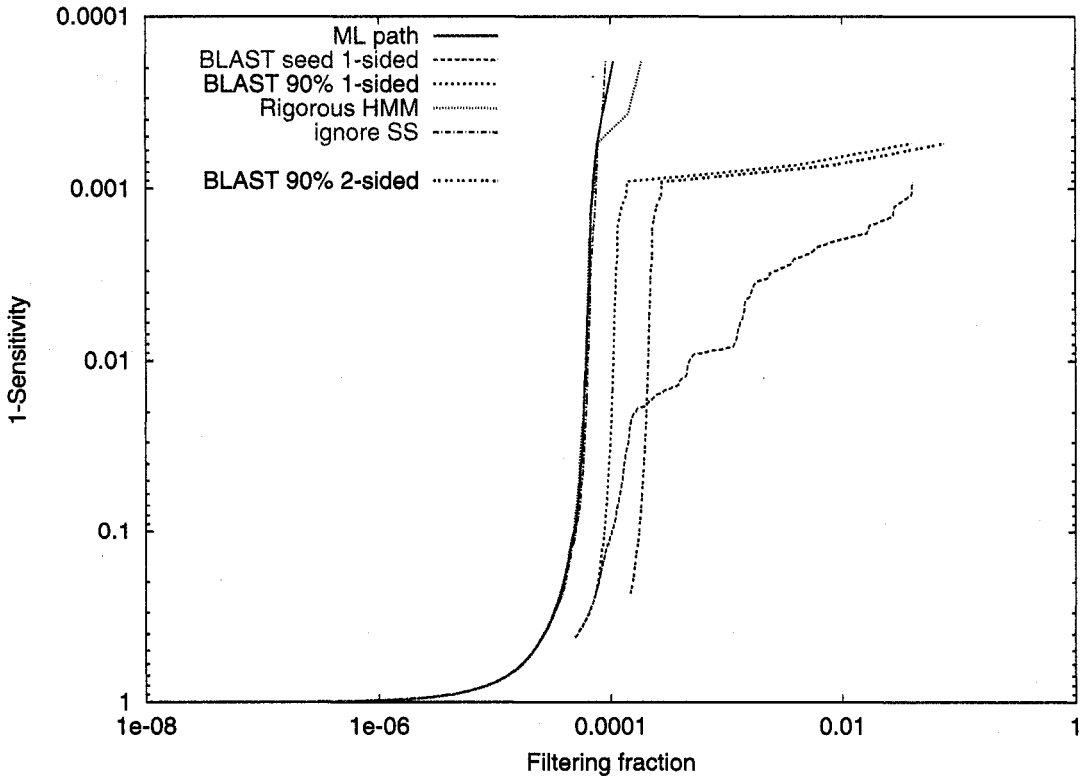


Figure 7.3: ROC-like curve: RF00001 (log-scale sensitivity)

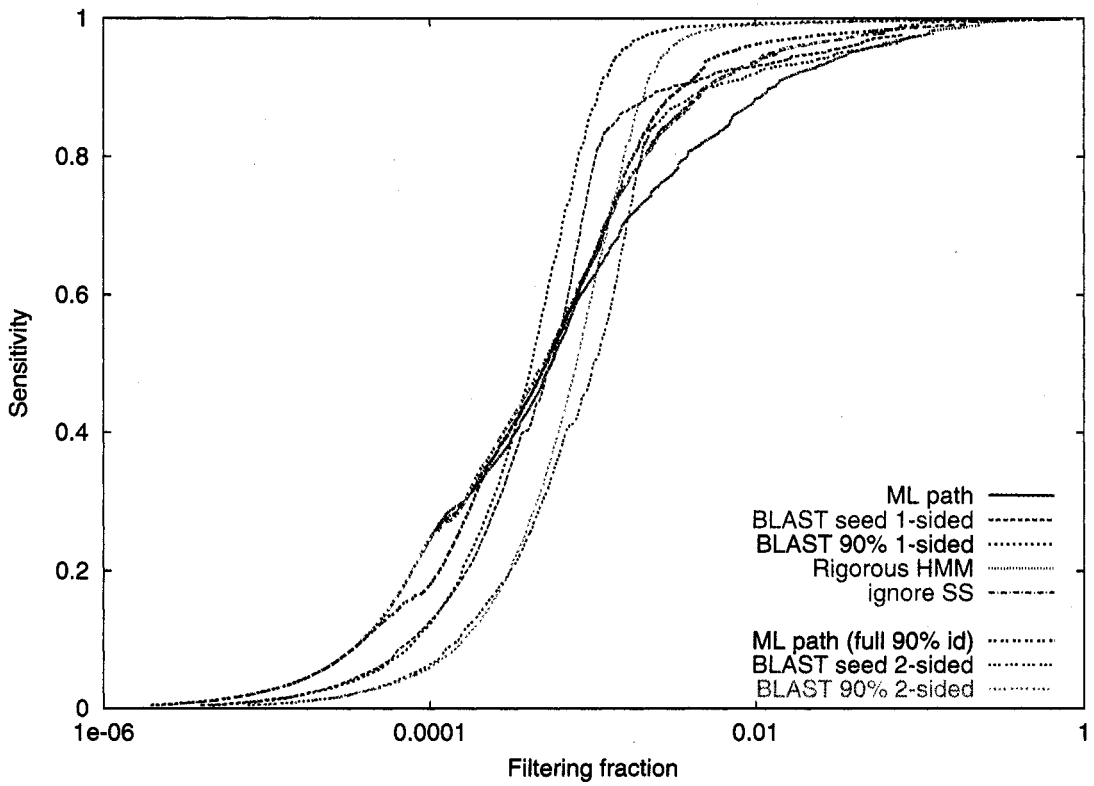


Figure 7.4: ROC-like curve: RF00005

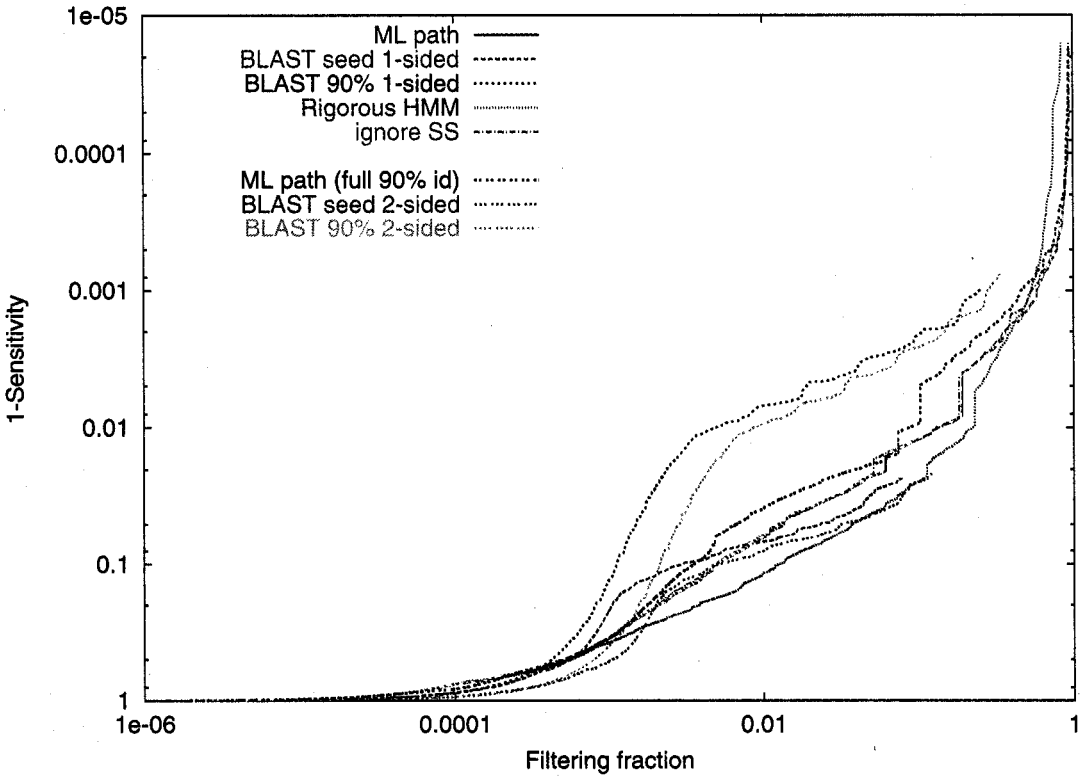


Figure 7.5: ROC-like curve: RF00005 (log-scale sensitivity)

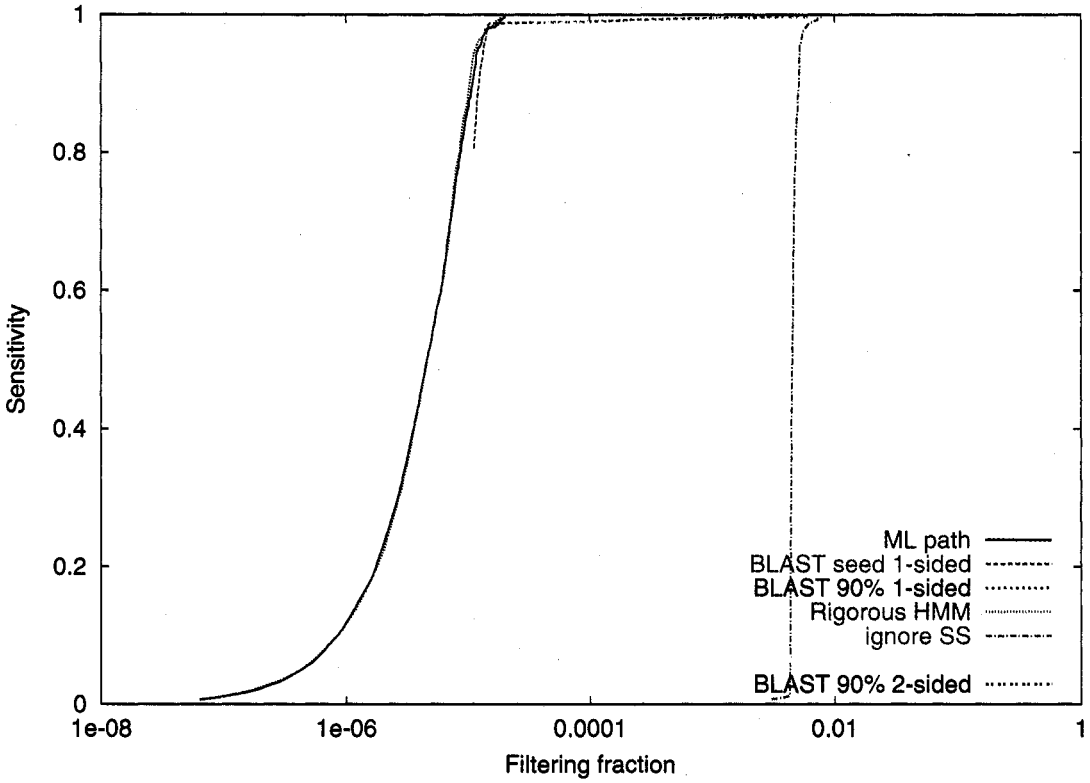


Figure 7.6: ROC-like curve: RF00010

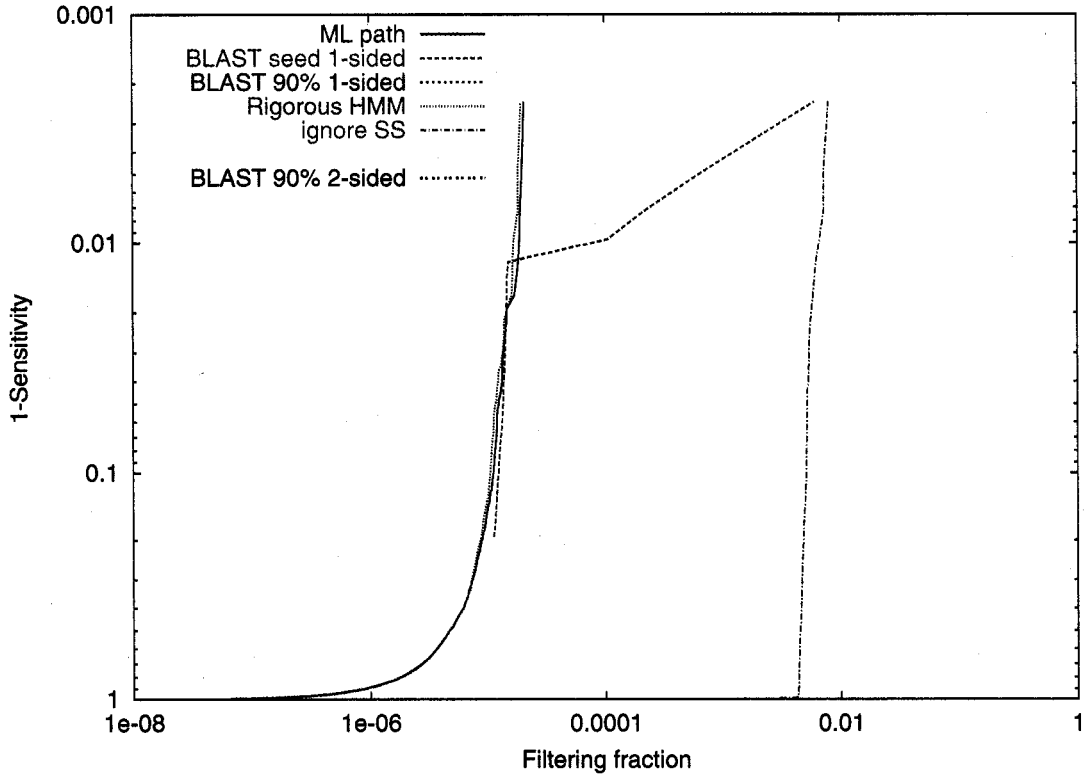


Figure 7.7: ROC-like curve: RF00010 (log-scale sensitivity)

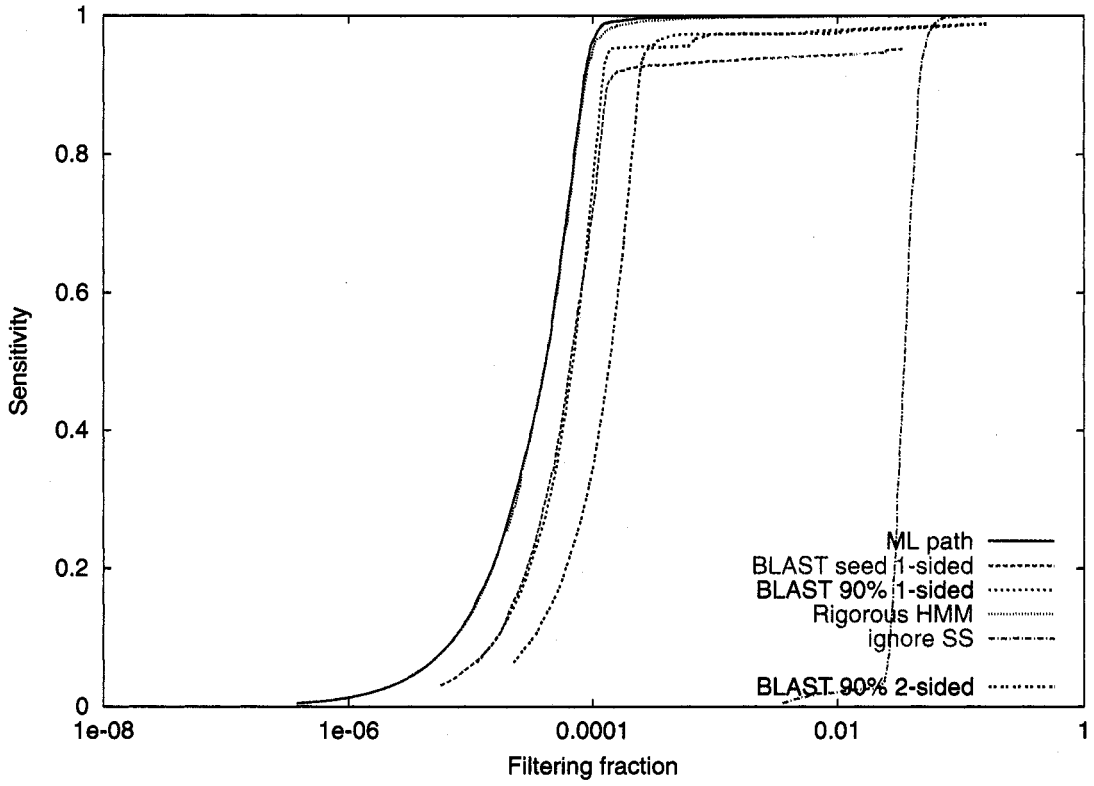


Figure 7.8: ROC-like curve: RF00029

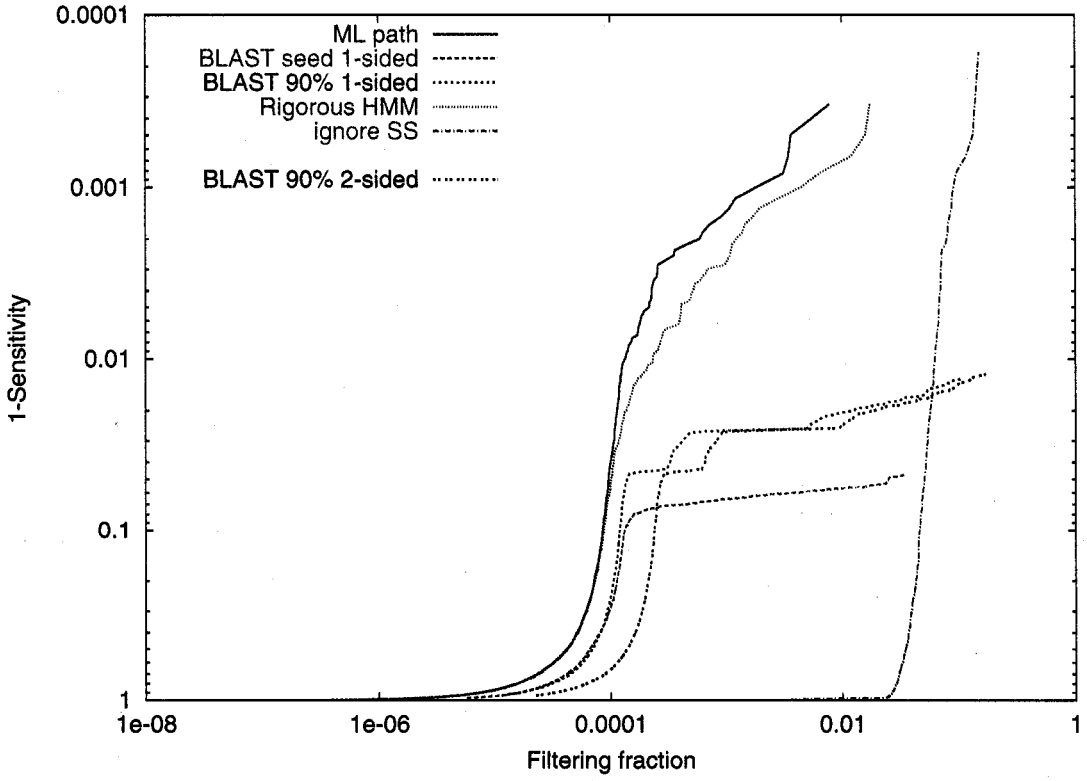


Figure 7.9: ROC-like curve: RF00029 (log-scale sensitivity)

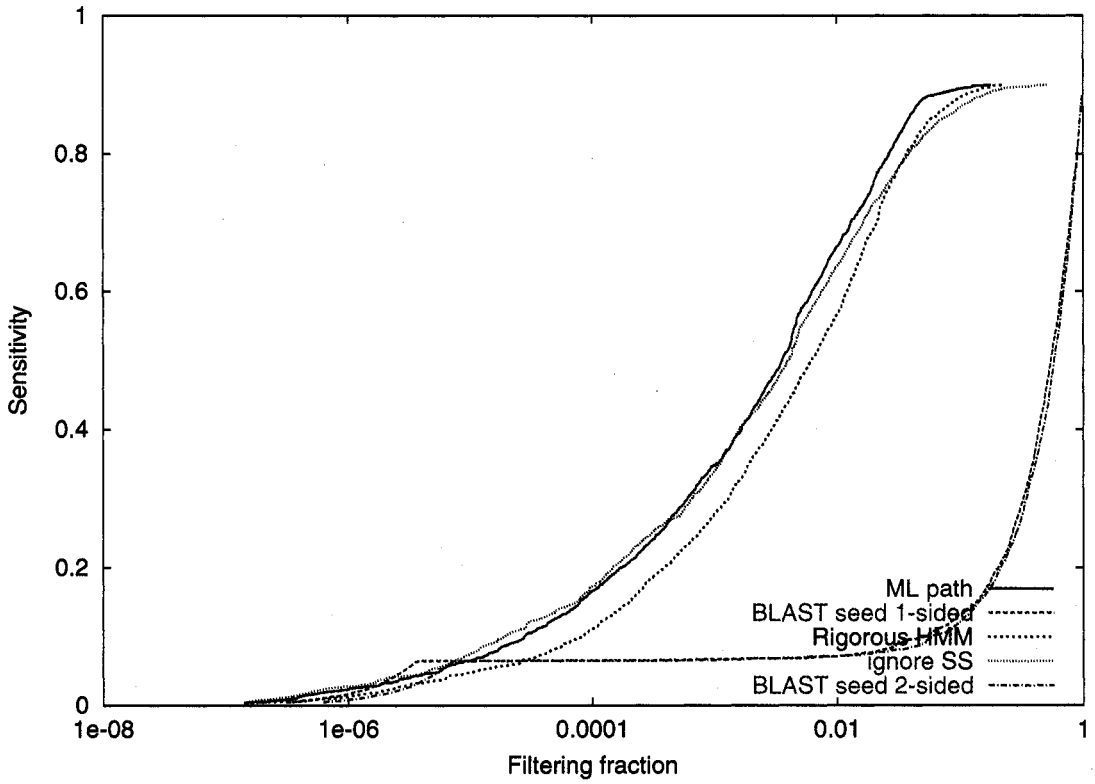


Figure 7.10: ROC-like curve: RF00031

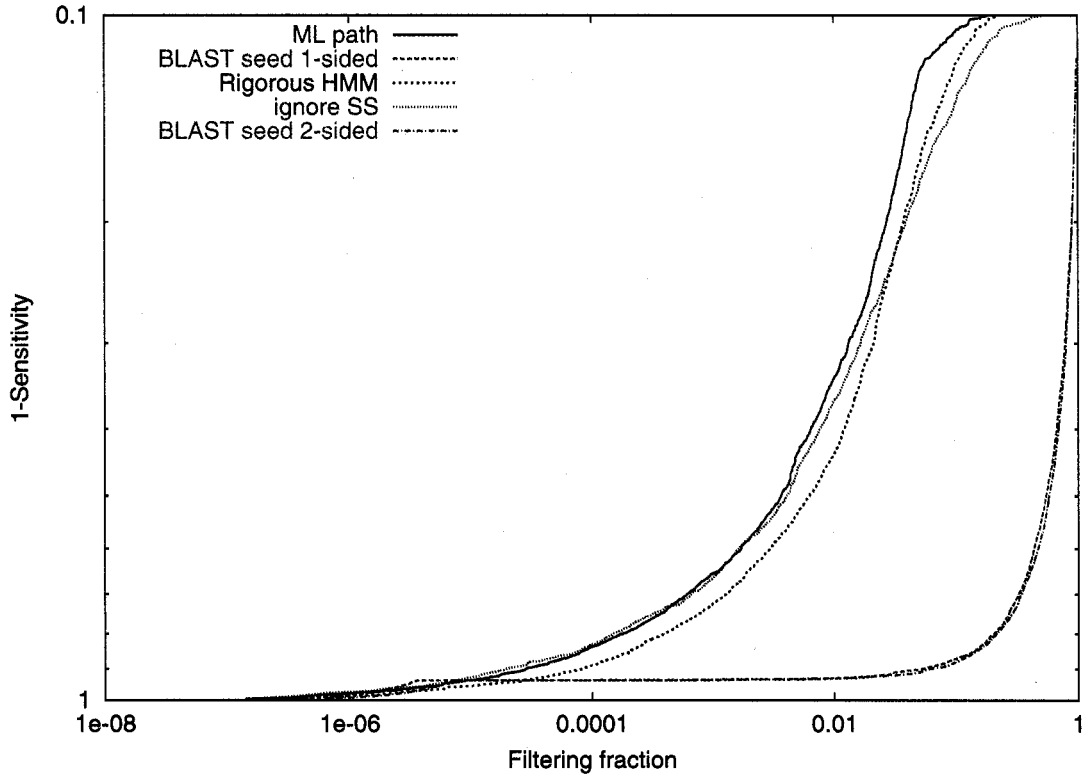


Figure 7.11: ROC-like curve: RF00031 (log-scale sensitivity)

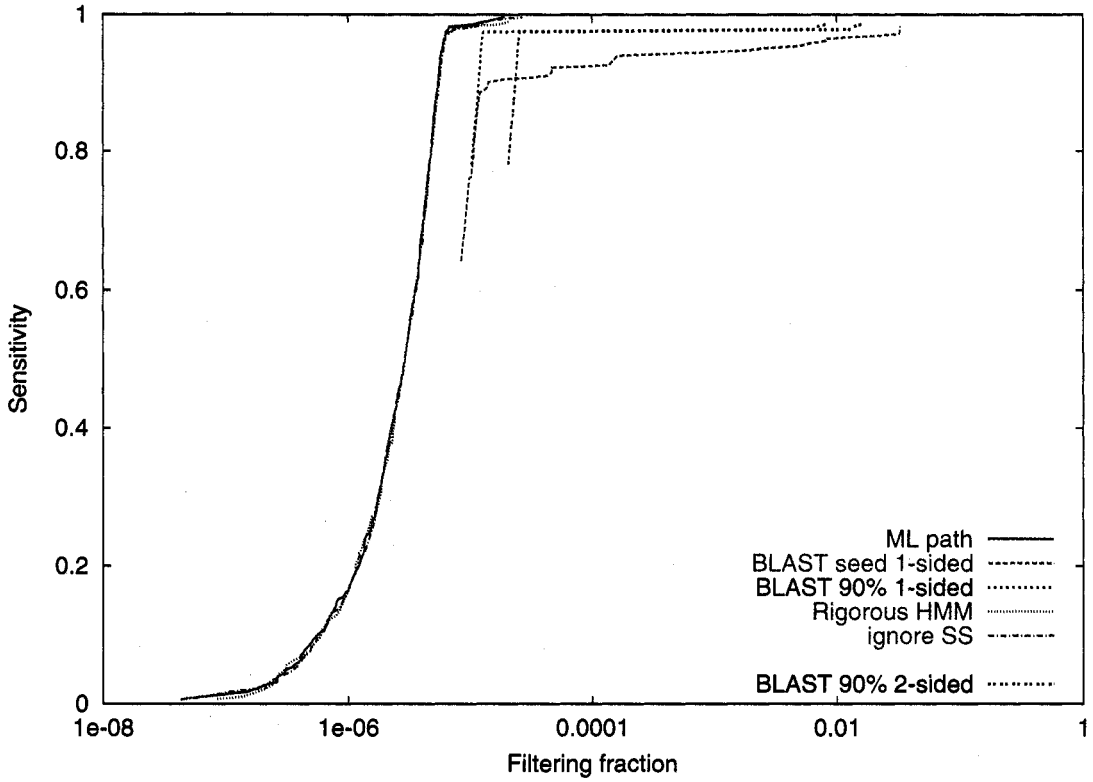


Figure 7.12: ROC-like curve: RF00059

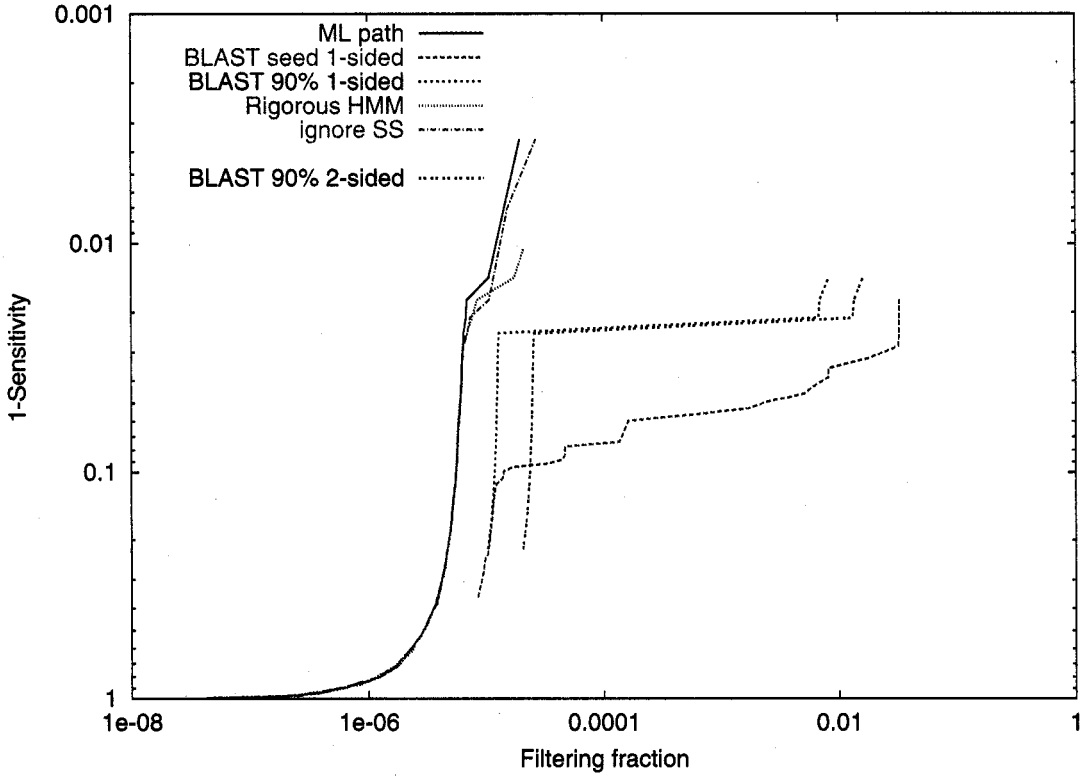


Figure 7.13: ROC-like curve: RF00059 (log-scale sensitivity)

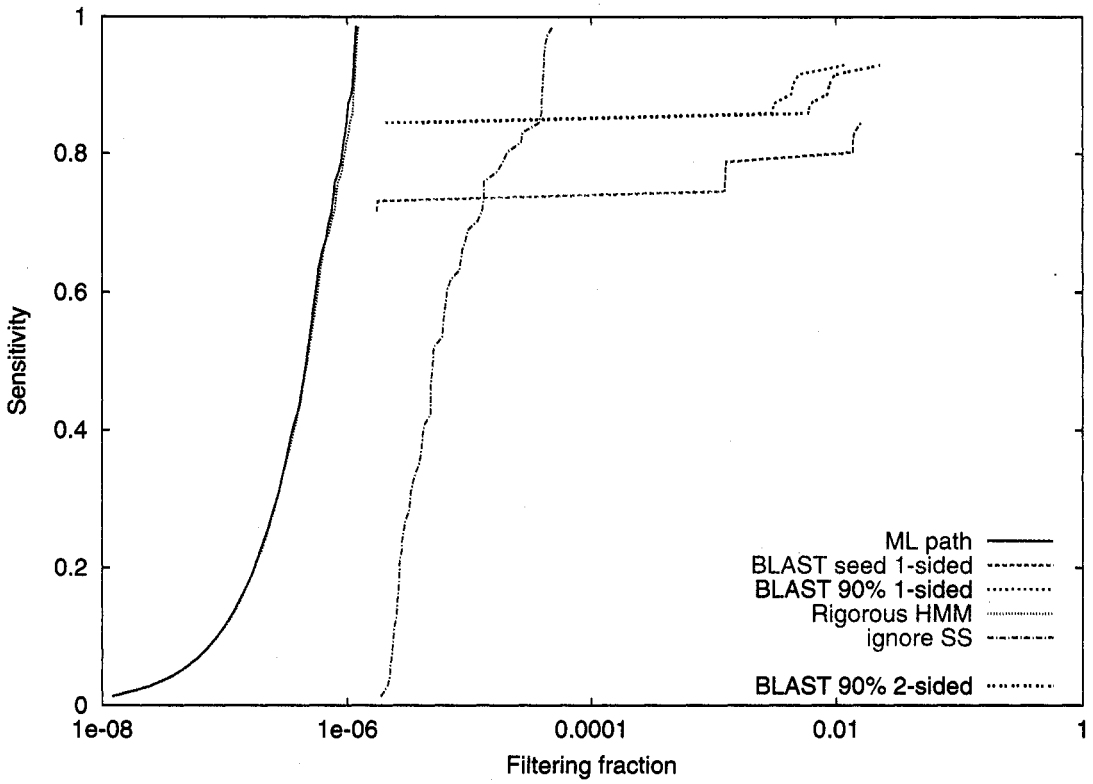


Figure 7.14: ROC-like curve: RF00168

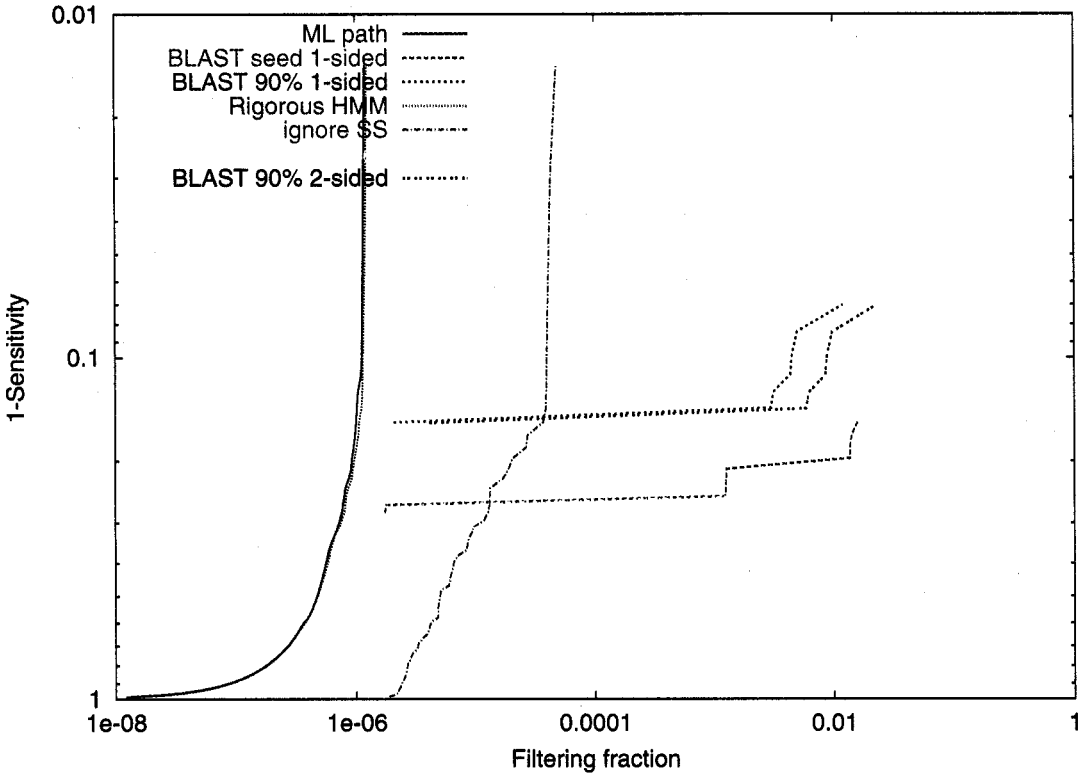


Figure 7.15: ROC-like curve: RF00168 (log-scale sensitivity)

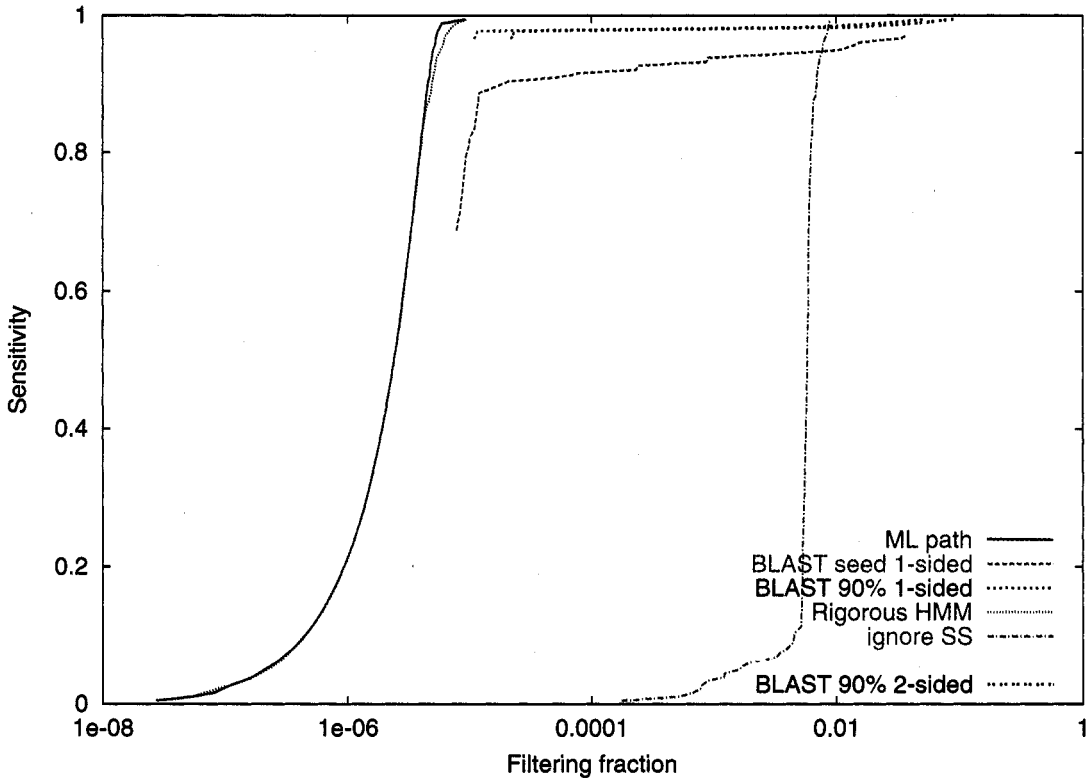


Figure 7.16: ROC-like curve: RF00174

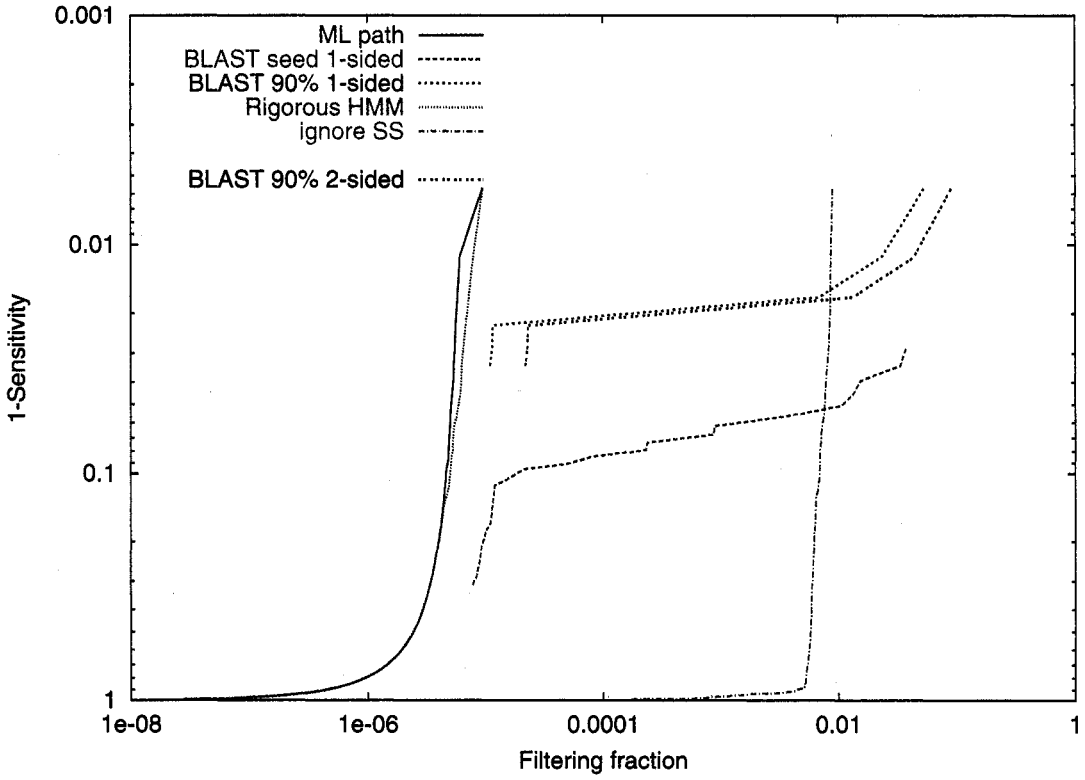


Figure 7.17: ROC-like curve: RF00174 (log-scale sensitivity)

Chapter 8

HEURISTIC HMMS FOR LOCAL CMS

In this chapter, I extend the ML-heuristic to local CMs, which accommodate missing or added domains with a fixed penalty (instead of several indel penalties) and thus can find more diverged family members that may have anomalous structures. (An example is tRNAs missing one arm of their classic cloverleaf structure.) Local CMs are used in ~40 Rfam database families.

Local CMs are also used by the RSEARCH tool [61], which finds highly diverged homologs from a single RNA sequence annotated with secondary structure, e.g., can find archaeal or *Arabidopsis* SRPs from a human SRP. RSEARCH uses local CMs with RNA score matrices to find highly diverged sequences from only one RNA; thus it is the first program, designed for RNA, to perform a task analogous to BLAST. Such searches are important when only one RNA is known, or when only a few highly correlated RNAs are known (which add little information beyond one sequence), to discover the true extent of the family.

To date, no methods to speed RSEARCH have been proposed. FastR [4] is related to CMs and uses structure-based filters; this program showed promising results on the 1.67-Mbase *A. pernix* genome and 1-Mbase synthetic sequences. I was unable to compare to this program, since the source code is not available, and since I wanted to test on large genome databases. BLAST has also been proposed as a filter for CM searches [48] in the context of Rfam, but it has poor sensitivity (see chapters 4, 5, 6 and 7)—and it is even worse for RSEARCH.

The CMs used with RSEARCH use arbitrary scores that were selected because they were found to yield the best sensitivity/specificity balance. This scheme is useful for the problem of finding RNA homologs, so it may be too restrictive to insist that a probabilistic scheme be used. Since the ML-heuristic is probabilistic in nature, I designed a general method to

convert a CM with arbitrary scores to a probabilistic model usable with the ML-heuristic.

I tested heuristic filters in two contexts: local Rfam database families and RSEARCH scans. ML-heuristic filters seem good heuristics, while BLAST often shows poor sensitivity.

To test filters for RSEARCH scans, I assembled 7 test sets with a known diverged family, and an arbitrary query RNA. Using current knowledge, I extracted the known true homologs, then asked how many RSEARCH could find with each heuristic filter. In this test, BLAST proves wholly inadequate, while the ML-heuristic generally achieves around 65-90% of RSEARCH's sensitivity. Using the ML-heuristic, scans are almost 100 times faster, yet are likely to find highly diverged homologs, if any.

Given these results, I used the ML-heuristic with RSEARCH in large-scale scans of phylogenetically constrained Rfam families, to find novel family homologs. I chose 147 bacterial, archaeal, eukaryotic and viral families in Rfam, selecting one member as an RSEARCH query using Rfam's RNA structure. I then scanned the 1.2-Gbase prokaryotic genome microbial RefSeq database [98] (for prokaryotic RNAs) the 0.54-Gbase RefSeq transcript (mRNA) database (for eukaryotic, *cis*-acting RNAs), the 8-Gbase RFAMSEQ (for eukaryotic, non-*cis* RNAs), RFAMSEQ's 0.25-Gbase viral subset (for viral RNAs). This huge workload is not feasible without filters.

For the 6S family, until recently known only in γ-proteobacteria, RSEARCH-based scans found homologs in distinct bacterial groups: δ- and β-proteobacteria (which diverged probably 2 billion years ago). Colleagues and I originally made this discovery using a more laborious process [6]. However, the test described in this chapter used no knowledge of 6S beyond what was known in γ-proteobacteria. (See Results.) I therefore conclude that RSEARCH with sequence filters is a powerful method for homology discovery, and could easily have made the discovery automatically.

In several of the other 147 families, RSEARCH finds homologs missed by Rfam's CMs—often by a significant margin. Probably most dramatic were the finding of T-boxes in δ-proteobacteria (only distantly related to the Firmicutes in which T-boxes are predominantly known) and the discovery that an RNA motif in Dengue virus is, in fact, present in a wide variety of flaviviruses.

8.1 Results

8.1.1 Evaluation

I evaluated BLAST, the local ML-heuristic and local rigorous HMMs (chapter 6) as heuristics. The evaluation used ROC-like curves, as described in section 7.1.1. However, in this section, I list the sensitivity at a filtering fraction of 0.01, i.e., one point on the ROC-like curve. (Complete graphs are in section 8.4.2.)

I first tested RSEARCH with filters. I performed tests from the RSEARCH paper: a human SRP query to find SRPs in 11 archaeal genomes, and (separately) in *Arabidopsis*, and a eubacterial SRP query on the 11 archaea. I also used a frog tRNA query on the 11 archaea. For larger-scale tests, I looked in RFAMSEQ for a *C. briggsae* 5S rRNA, and in a 1.2-Gbase bacterial genome database for *E. coli* 6S and a *B. subtilis* lysine riboswitch. (Non-SRP queries were derived from Rfam.) These tests represent a mix of RNA sizes and structures, and extremely diverged tests (e.g., human to archaea) compared to large scale screens (on RFAMSEQ and bacterial genomes).

To measure filter sensitivity, I inferred positives of 6S using nucleotide coordinates of 6S RNAs from recent work [6]. I then ran RSEARCH at an E-value cutoff of 10 around these coordinates (padding 300 nucleotides on either end). I assume that these are true positives that RSEARCH could detect with a filter of 100% sensitivity. For other families, I scanned using Rfam's CMs (i.e., the whole family, not just the single query RNA) with non-local rigorous filters (chapter 5), running RSEARCH around these hits as before.

Each RSEARCH test in Table 8.1 uses a query sequence from an Rfam family or SRP from the RSEARCH paper. This sequence is annotated with curated secondary structure (from Rfam or the RSEARCH paper), forming a valid RSEARCH query. I run this RSEARCH query with a filter: BLAST, ML-heuristic or rigorous. The heuristics were tested relative to the hits found with rigorous scans. Table 8.1 shows sensitivity at fraction 0.01, which yields scans almost 100 times faster.

BLAST itself is faster than an HMM. However, at the higher filtering fractions often required for good sensitivity, overall scan time is dominated by the CM. In sensitivity, BLAST is generally the worst method on Rfam searches, but is often competitive. For RSEARCH

Table 8.1: Sensitivity at filtering fraction 0.01. Top lines used Rfam's CMs; the rest used an RSEARCH CM made from one family member. (RSEARCH databases: [1]1.2 Gbase bacterial, [2]RFAMSEQ, [3]11 archaeal genomes from RSEARCH paper, [4]*Arabidopsis*. *H. s*=human, *B. s*=*B. subtilis*.) The family's average length is given, then sensitivity (relative to the CM, from Rfam or RSEARCH) of BLAST, ML-heuristic and rigorous HMMs at filtering fraction 0.01. Last is speedup ratio of the ML-heuristic at filtering fraction 0.01 versus raw CM, extrapolated from small test sequences. (Filtering fractions other than 0.01 appear in section 8.4.2.)

Family	**avg**	**% sensitivity**			**speedup**
	len	**BLAST**	**ML**	**Rig.**	CM/ML
Tests on local Rfam families					
5.8S, RF00002	142	99.4	100	100	47×
Vault, RF00006	103	100	100	100	47×
6S, RF00013	173	100	92.9	100	141×
CsrB, RF00018	240	100	100	100	102×
HgcC, RF00062	169	100	100	100	68×
U15, RF00067	115	85.4	100	100	29×
QUAD, RF00113	139	83.9	100	100	48×
S15 UTR, RF00114	162	38.2	100	100	60×
5' signal, RF00171	143	100	100	100	54×
Tests on RSEARCH queries					
6S, RF00013[1]	173	30.3	87.8	84.8	75×
5S, RF00001[2]	115	76.4	80.5	78.9	52×
tRNA, RF00005[3]	71	0.3	21.6	20.3	31×
lysine, RF00168[1]	181	4.3	67.4	82.6	50×
H.s SRP[4]	298	0	70	0	107×
H.s SRP[3]	298	0	50	0	107×
B.s SRP[3]	98	16.7	66.7	83.3	103×

Table 8.2: Summary of RSEARCH homology searches. RSEARCH searches were done from an arbitrarily chosen family member on 147 families in Rfam 7.0 (or Rfam 6.1). For 57 families RSEARCH found homologs with E-values less than 1 (borderline significance) that were not found by Rfam's CM. The table classifies these in two dimensions: evidence that homologs are true (Strong = clear supporting annotation and plausible; Plausible/unclear = no evidence either way, or conflicting evidence; Weak/dubious), and how diverged the hits are (Significant = homologs found in new upper-level taxonomy groups, or highly diverged species in the same group; Moderate = species that are not in the same genus; Modest = species that are not new, or highly similar to an existing species). Each cell in this table counts the number of applicable families in Rfam.

		Evidence		
		Strong	Plausible/unclear	Weak/dubious
Divergence	Significant	10	4	12
	Moderate	8	7	0
	Modest	10	4	2

scans, BLAST is always worst, often overwhelmingly. This may be because RSEARCH uses only one RNA (unlike Rfam families), which is hard for BLAST to generalize from.

On the Rfam families, the ML-heuristic and rigorous HMMs are comparable, with rigorous HMMs sometimes better. There are some filtering fractions other than 0.01 where the ML-heuristic wins even in these cases, so the comparison is not so clear. The ML-heuristic is much quicker to create (1 sec vs. hours or days), so may be preferable. For RSEARCH queries, the ML-heuristic seems overall slightly better than rigorous HMMs. With the exception of the RF00005 (frog tRNA), the ML-heuristic is able to realize most of the sensitivity of RSEARCH, in a fraction of the time. This is an important practical contrast to prior solutions: using RSEARCH is more sensitive, but too slow to use on a large scale, while using BLAST as a filter achieves little of RSEARCH's sensitivity.

8.1.2 Novel homologs found with filtered RSEARCH

I applied RSEARCH with the ML-heuristic to find novel RNA homologs. RSEARCH fills an important gap in tools for ncRNA research: finding homologs from a single instance.

When new RNAs are discovered, often only one instance is known, so standard CMs can be problematic. RSEARCH uses local CMs and a scoring matrix designed to discover diverged homologs, not just the immediately apparent ones. Therefore, RSEARCH can be useful to break outside of cases where only a limited number of largely redundant RNAs are known in a family. A key limitation has been RSEARCH's exorbitant CPU needs. The previous section showed that ML-heuristic filters realize most of RSEARCH's sensitivity in a fraction of the time. For example, 16 of the bacterial RNA scans (below) alone consumed 51 CPU days. If RSEARCH averages 60 times slower, it would take 8 CPU years.

I chose 147 Rfam families that were phylogenetically restricted, in bacteria, archaea, eukaryotes and viruses. I ran RSEARCH with the ML-heuristic at E-value cutoff 100, but was most attentive to E-values below 1. For bacterial scans I used full and partial genomes in the RefSeq database [98], totaling 1.2 Gbases. Eukaryotic RNAs were scanned on the ~8-Gigabase RFAMSEQ, except that, to reduce false positives, eukaryotic *cis*-regulatory RNAs were scanned on the ~540-Mbase RefSeq database of transcripts (mRNAs). viral RNAs used the ~250-Mbase viral subset of RFAMSEQ. The scale of this task indicates the large-scale scans that filters make possible. Moreover, scans uncovered new, plausible homologs in several cases. In view of these results, I believe filtered RSEARCH is an important addition to a suite of tools for RNA research.

For each family where filtered RSEARCH found homologs not in Rfam, I classified how diverged the homolog was to the known family members, and evaluate evidence that the homologs are real; Table 8.2 summarizes. Detailed information on all families scanned is at section 8.4.1. The following sections discuss highlights.

6S. Recent work shows that 6S, previously known in γ-proteobacteria, is not only found outside γ-proteobacteria [133, 123] but is in fact found in most bacterial groups [6]. This chapter's techniques could have instigated this discovery: I extracted the *E. coli* member of the 6S alignment in Rfam 6.1 for an RSEARCH query. This alignment was created before anyone knew of 6S's presence beyond γ-proteobacteria, so uses no knowledge of 6S's widespread nature. Moreover, this experiment is identical to the ones I performed on other Rfam families (below): I automatically extracted an arbitrary member of an Rfam family,

and scanned an appropriate database. I did this experiment before doing any sensitivity tests (Section 2.1), and got the results below on the first try.

Using the ML-heuristic, I found now-known 6S homologs in δ-proteobacteria (*Geobacter sulfurreducens*) and β-proteobacteria (*Nitrosomonas europaea*, *Chromobacterium violaceum* and *Bordetella*), with E-values less than 1 (two of these hits have E-values < 0.05). γ- and β-protebacteria have diverged $\sim$2 billion years ago [8], so the homologs are highly diverged. (I found no divergence time estimates for δ-proteobacteria.) RSEARCH can also find cyanobacterial 6S, but the ML-heuristic misses them, so a more sensitive filter would be even better. However, the search required a practical amount of computer time (8 CPU days). Since this experiment was fully automated, used only γ-proteobacterial-based knowledge, and required practical computer resources, my program could have discovered that 6S exists out of γ-proteobacteria if it had been used.

T-box. I discovered T-boxes (Rfam ID RF00230) [50] in two *Geobacter* species, which are Gram-negative δ-proteobacteria. This is significant, because the Rfam T-box family consists only of Gram-positive species (Firmicutes and one Actinobacteria species). Although current thinking is that Firmicutes are more closely related to δ-proteobacteria than to any other proteobacterial group, they are classified into separate descriptive groups and evolutionarily are deeply divergent.

T-boxes regulate the expression of certain amino acid synthesis operons as part of their 5' UTR. For example, a leucine-dependent T-box contains a leucine codon. If leucine is depleted, uncharged leucyl-tRNAs are abundant, so bind this codon, and change the mRNA's structure, affecting expression. The candidate *Geobacter* T-boxes are upstream of *leuA* (leucine biosynthesis) genes, and their "specifier" codons correspond to leucine. In addition to the homology detected by RSEARCH, this provides compelling evidence of functional T-boxes in a highly diverged species. Moreover, this discovery was detected by a filtered RSEARCH query without any prior knowledge of it. (Subsequent to this discovery, a colleague found a prior claim of a *Geobacter* T-box [136]. Their discovery strategy involved scanning only UTRs likely to contain T-boxes. My scans found the diverged T-boxes when run on whole genomes, which will be more practical for RNAs that are not *cis* elements.)

RNase E element: RSEARCH discriminates better than Rfam's CMs. I found an RNase E 5' element (RF00040) in *Photorhabdus luminescens*, a γ-proteobacteria diverged from the previously known (γ-proteobacterial) family members. RSEARCH reports a dramatic E-value of 10^{-16}, yet the Rfam model scores it as 15 bits below the threshold. This hit is 5' to an RNase E gene, so can be assumed a true homolog.

Threonine operon leader: transcription terminator loops. For the threonine operon leader (RF00506), known only in enterobacteria, RSEARCH predicted homologs in a wide variety of bacteria. Unfortunately, the local begin rule was used to match a part of the leader that is a transcription terminator loop. Terminator loops appear in a variety of contexts in bacteria, so are a confounding factor for local models.

BiP IRES: truncated RNA. The Internal Ribosome Entry Site (IRES) is an mRNA structural element that is an alternate means to initiate translation in eukaryotes and viruses The BiP IRES upregulates translation in heat shock. The Rfam family (RF00223) has only human members. RSEARCH predicted a homolog in a BiP gene mRNA in mouse, which Rfam's CM did not find. The mouse mRNA may be partial, and Rfam's non-local CM should have trouble matching only part of the RNA. There are several other cases of RSEARCH finding possibly truncated RNAs missed by non-local CMs (see section 8.4.1).

3' UTR viral element. Family RF00497 is a conserved element in strains of Dengue virus. The Dengue virus genome is essentially a single, ~11,000-nucleotide mRNA molecule with 5' UTR, ORF and 3' UTR. Its ORF encodes a polyprotein that is processed into about 10 proteins. In the 3' UTR is a conserved structural element that is thought to play a role in viral replication. The Dengue virus is carried by mosquitoes and infects humans. Viruses with mutant 3' UTR elements replicate poorly in monkey cell culture, and some mutants cannot replicate at all in mosquito cells [138].

Filtered RSEARCH predicted homologs in several types of flavivirus, the taxon containing Dengue viruses: Japanese encephalitis virus, West Nile virus, Kunjin virus, Murray Valley encephalitis virus and Usutu virus. Not only were these hits in other flaviviruses,

but they were all in the 3' UTR. I integrated the new sequences into an improved MSA for the family, and did scans with non-RSEARCH CMs. With the head start given by the RSEARCH hits, I was able to find homologs in essentially every sequenced flavivirus that is known to infect a mammal and be carried by mosquito or tick. (Some flaviviruses have no known vector, while others infect only mosquitoes and not mammals.) I added Yellow Fever, Tick-borne encephalitis, Langat virus, Louping ill virus, Powassan virus and the intimidating Omsk Hemorrhagic Fever virus. Moreover, these results turned out to intersect members of RF00185, and unified these two related families. (Indeed, RSEARCHes from RF00185 turned up several homologs outside of the Japanese encephalitis group it was known in, including Dengue virus.)

Summary. The ML-heuristic filter sacrifices some sensitivity, but its speed is crucial to make scans practical. Its sensitivity is enough to find highly diverged RNAs, most notably 6S outside of γ-proteobacteria and T-boxes outside of Gram-positive bacteria. (And with smaller databases, carefully chosen to eliminate approximately redundant or overly diverged sequences, sensitivity may be even better.) Thus, we now have a practical tool to find diverged RNAs when only one or a few, phylogenetically constrained members are known.

8.2 ML-heuristic for local CMs

The ML-heuristic was described, in the non-local case, in chapter 7. Recall that a CM is a generative model, so we can sample from its distribution a random multiple-sequence alignment (MSA) of n sequences. As $n \rightarrow \infty$, the MSA captures the distribution more perfectly. Since profile HMMs are probabilistic models, we can train the HMM on the CM's infinite MSA to create the ML-heuristic.

The ML-heuristic can be generalized to the local case by considering local begins and local ends to be elements in an alignment. In the context of the ML-heuristic, MSAs are effectively viewed as a set of CM parses, since they unambiguously define, for each sequence, what nucleotides are inserted/deleted and which nucleotides match which MSA positions. Therefore, they unambiguously indicate a specific CM parse. When a local CM creates a random MSA, this MSA should indicate where local begins and local ends were used for

each sequence in the MSA.

These local elements are mapped to profile HMMs in a manner similar to the rigorous local case (chapter 6):

- CM's local end rule $S_i \rightarrow \epsilon S_N$ maps to HMM rules $\overline{S}_i^L \rightarrow \epsilon \overline{S}_i^E$ and $\overline{S}_i^E \rightarrow \overline{S}_i^R$. As with the rigorous case, the ML-heuristic has no need to treat the two HMM rules separately, since they always go together.

- For local begins, CM rule $S_0 \rightarrow S_i$ maps to $\overline{S}_0^L \rightarrow \overline{S}_i^L$ and $\overline{S}_i^R \rightarrow \overline{S}_0^R$. As in the rigorous case, the two HMM rules do not always go together in HMM parses, since the profile HMM cannot enforce consistency between left and right nucleotides. However, in CM parses there is only one local begin rule; whenever the CM's random MSA produces a local begin in a sequence, this maps to both left and right local begins in the HMM. Therefore, the expected use of the $\overline{S}_0^L \rightarrow \overline{S}_i^L$ rule is the same as the expected usage of the $S_0 \rightarrow S_i$ rule in simulating the infinite MSA. And the same with the $\overline{S}_i^R \rightarrow \overline{S}_0^R$ rule.

8.2.1 Converting arbitrary CMs to probabilistic models

Unfortunately, RSEARCH CMs are not probabilistic. A probabilistic model defines probabilities for each parse such that (1) each probability is in the interval $[0, 1]$ and (2) the probabilities of all parses sum to 1. Given such a model, it is possible to randomly sample parses for use in the random MSA.

RSEARCH CMs define scores of parses. These scores are typically not in $[0, 1]$ and can even be negative. They do not sum to 1. Therefore, RSEARCH CMs cannot be used as generative models.

Moreover, an RSEARCH CM does not define a probabilistic interpretation of its scores. In traditional CMs, scores are a log odds ratio, i.e., the score $s = \log_2(p_{\text{CM}}/p_{\text{null}})$, where p_{CM} is the probability according to the CM, and p_{null} the null-model-assigned probability. The p_{CM} define a probabilistic model.

By contrast, RSEARCH CMs use scores that were chosen because they led to lower error rates in empirical tests than other numbers. These numbers have no direct probabilistic

interpretation. They are used for transitions and for local begin/end probabilities. (The scores for emissions do have a probabilistic interpretation.) Although it would be convenient for our present purpose to re-design RSEARCH to be fully probabilistic, the choice of numbers that minimize error rates is a reasonable one, and it may be difficult to generate training data that would help learn local begin/end scores. Therefore, I believe it is useful to have a general method to convert non-probabilistic CMs to probabilistic ones.

Definition of simple CMs with arbitrary scores. I define a simplified CM model (essentially that of chapter 3) appropriate for schemes like RSEARCH's with arbitrary scores. All rules are of the form $\mathrm{S}_i \rightarrow x_L \mathrm{S}_j x_R$. ($j$ is used here instead of $i+1$ to accommodate local begin/end rules, which are not consecutive states.) In the simplified case, $j > i$, i.e., insertion state self loops are not allowed ($j \neq i$). They will be addressed later. End rules $\mathrm{S}_i \rightarrow \epsilon$ are allowed.

All rules have arbitrary scores. The score of rule r is denoted $S(r)$. For a parse π if $R(\pi)$ is the sequence of rules in π, the parse's score is $S(\pi) = \sum_{r \in R(\pi)} S(r)$; this is the standard definition.

To score a sequence, we use the score of its Viterbi parse.

Assignment of rule probabilities. I define the *pseudo-probability* of rule r, $p_r = 2^{S(r)}$. Intuitively, this is motivated by the observation that additive rules (like $S(r)$) are similar to logarithms, so exponentiating them converts to a multiplicative space. Indeed, later I show that these numbers behave like probabilities. The choice of base 2 is also discussed later. I define the pseudo-probability of parse π to be $p_\pi = \prod_{r \in R(\pi)} p_r$, with $R(\pi)$ being π's rules as before.

To be probabilistic, the sum of p_r of rules with the same left-hand side (S_i) must sum to 1. I use a recursive algorithm to achieve this.

Let $N(\mathrm{S}_i)$ be a *normalizing constant* for the sub-CM rooted at state S_i. Recursively sum over rules r with S_i in the left-hand side:

$$N(\mathrm{S}_i) = \sum_{r = \mathrm{S}_i \rightarrow x_L \mathrm{S}_j x_R} p_r N(\mathrm{S}_j)$$

For a base case, rewrite all rules $\mathrm{S}_i \rightarrow \epsilon$ as $\mathrm{S}_i \rightarrow \mathrm{S}_\infty$ for new state S_∞. $\mathrm{S}_\infty \rightarrow \epsilon$ is its only rule, and $N(\mathrm{S}_\infty) = 1$.

I now induce true probabilities p'_r: for rule $r = \mathrm{S}_i \rightarrow x_L \mathrm{S}_j x_R$,

$$p'_r = \frac{p_r N(\mathrm{S}_j)}{N(\mathrm{S}_i)}$$

This model with p'_r is probabilistic. Due to the normalizing $N(\mathrm{S}_i)$, the sum of probabilities of rules leaving state S_i sum to 1. Thus, the new model with p'_r is a conventional, probabilistic SCFG.

Probabilistic scheme agrees with Viterbi scores. We can define scores of rules based on the probabilistic transformation, $S'(r) = \log_2 p'_r$. This is the conventional way to translate probabilities into additive scores. In this case, we see that these scores are an affine transformation of the original arbitrary scores $S(r)$.

To see this, assume rule $r_2 = \mathrm{S}_j \rightarrow x_L \mathrm{S}_k x_R$. Then

$$p'_{r_2} = \frac{p_{r_2} N(\mathrm{S}_k)}{N(\mathrm{S}_j)}$$

So, if r and r_2 are part of a parse, we see that $N(\mathrm{S}_j)$ cancels out in

$$p'_r \times p'_{r_2} = \frac{p_r p_{r_2} N(\mathrm{S}_k)}{N(\mathrm{S}_i)}$$

In a full parse, we are left with

$$p'_\pi = \frac{N(\mathrm{S}_\infty) \prod_{r \in R(\pi)} p_r}{N(\mathrm{S}_0)} = \frac{p_\pi}{N(\mathrm{S}_0)}$$

and $N(\mathrm{S}_0)$ is the full normalizing constant ($N(\mathrm{S}_0) = \sum_\pi p_\pi$). Taking base 2 logarithms of both sides, we obtain

$$S'(\pi) = \log_2 \left(p_\pi / N(\mathrm{S}_0)\right) = S(\pi) - \log_2 \left(N(\mathrm{S}_0)\right)$$

If we instead use $\log_b$, we obtain $(\log_b 2)\, S(\pi) - \log_b \left(N(\mathrm{S}_0)\right)$.

Thus, in terms of scores, the transformed model uses an affine transformation of the original scores. Therefore, the scores preserve ordering—the original model (with arbitrary scores) and the transformed model will rank sequences in the same way. Indeed, if $\log_2$ is used, the difference between scores is preserved.

Infinite normalizing constants. With arbitrary scores and insert state self loops, the normalizing constant $N(S_0)$ can be infinite. Fortunately, this possibility does not arise in practice. There are two extant flavors of CM. Although Rfam (traditional) CMs usually have infinite normalizing constants, they have a probabilistic interpretation already.

RSEARCH CMs are not fully probabilistic, but fortunately their scores are such that the normalizing constant is finite in the above scheme. (Emission scores for each nucleotide in insertion states are zero, since CM and background models have the same probability in this case. There are 4 nucleotides to sum over. The self-loop score in RSEARCH is -5. $4 \times 2^0 \times 2^{-5} = 1/8 < 1$, so the normalizing constant is finite.) Therefore, there is no immediate need to treat infinite normalizing constants.

However, I speculate that the following scheme would be appropriate for the ML-heuristic. The scheme is motivated by the following observation. In traditional CMs, odds ratios are used. If we multiply by the null model (whose probabilities are less than 1), we obtain a probabilistic CM. The ML-heuristic can be applied to this CM to make an HMM. We may then divide by the null model, thus using odds ratios in the HMM. Observe that the null model supported multiplying emission pseudo-probabilities by constants less than 1; with low enough constants, the normalizing constant is finite.

In general, this scheme divides all emission CM pseudo-probabilities by some constant k, then applies the ML-heuristic, and then multiplies k into emission pseudo-probabilities of the HMM. If $k = 4$, the number of nucleotides, the scheme corresponds to removing/adding a uniform 0th-order null model.

Since normalization occurs, this scheme is probabilistic, as before. However, longer sequences are penalized more than shorter ones, since we divide by k for each emission. Therefore, the ordering of sequences by scores is violated, although within a given length, the order is not violated. Since CMs and HMMs define emissions in similar ways, it seems reasonable to suppose that perturbing the ordering of different-length sequences in the CM, then doing the reverse operation in the HMM, preserves the essential information in the CM. However, I have not investigated this formally.

Relationship to statistical mechanics. The transformation (in the finite case) resembles *partition functions* used in statistical mechanics [16]. For example, RNA structure prediction programs like Mfold and RNAfold use partition functions to define probabilities of the different predicted structures [88]. If $E(\pi)$ is the free energy of the structure corresponding to parse π, then the probability of a structure is defined as:

$$Pr(\pi) = \frac{e^{-E(\pi)/k_B T}}{Z}$$

where k_B is the Boltzmann constant, T is the temperature and Z is the partition function, a normalizing constant equal to $\sum_\pi e^{-E(\pi)/k_B T}$. Suppose $S(\pi)$ is π's arbitrary score, and we set T such that $-1/k_B T = \ln 2$. Then the scores exactly correspond to free energies in the context of RNA structure prediction, and have the same probabilistic interpretation. (In this context, using bases other than 2 corresponds to changing the temperature T.)

Summary. The complete transformation algorithm is as follows. We are given a CM with arbitrary scores (i.e., an RSEARCH CM). We convert to pseudo-probabilities. Then we convert the pseudo-probabilities to a probabilistic model by normalizing. Given this probabilistic model, we apply the ML-heuristic. The net result of this transformation is an ML-heuristic HMM. It exhibited good performance in our experiments of section 8.1.1, although it is possible that another transformation scheme would perform even better.

The case for conventional Rfam CMs, local or not, is the same as the previous chapter. We extract the probabilistic CM by removing the null model, and apply this null model to the HMM.

8.3 Discussion

The use of RSEARCH shows promise at expanding certain families. Although the ML-heuristic sacrifices a significant amount of RSEARCH's sensitivity, most of the sensitivity remains, and searches run in practical time. BLAST sensitivity is much worse in the context of RSEARCH.

Chapter 10 discusses in more detail two families expanded in this chapter: RF00460 and RF00497.

8.4 Additional details

8.4.1 Complete results from RSEARCH scans

In this section, I describe the results of all 57 RSEARCH scans where a hit was found that is not in Rfam. For each family, I state its Rfam ID, and give a brief description, the taxa in which the Rfam members are found, the taxa where new hits are found and analysis of the biological plausibility and significance of the new hits. I note the following details:

- All hits mentioned are those that Rfam's model fails to find, but an RSEARCH scan finds. For example, there are some mentions of *Erwinia*, and this is not simply because it is a new genome in the database.

- RSEARCHes on Rfam IDs less than RF00382 were done on Rfam version 6.1. Rfam IDs RF00382 and higher were done on Rfam version 7.0. Since Rfam is constantly being updated, it is possible that discoveries in this dissertation have already been added to Rfam.

- Some hits with marginal E-values (~0.5) were not discussed when they were obvious false positives.

- In some families there are both highly diverged hits with weak supporting evidence as well as less-diverged hits with strong evidence. I have generally classified the families under the latter (stronger evidence), but mention both classes of hits in the text.

- Judgement of phylogenetic distance is not generally based on any objective study, but rather a subjective judgement based on the taxonomy tree maintained at the NCBI web site (`www.ncbi.nlm.nih.gov`) and my personal experience of how diverged RNAs usually are.

- Additional information on families is available by their Rfam ID number at `rfam.wustl.edu`.

Families expanded by RSEARCH: significant expansion and strong evidence

RF00013 6S, ncRNA gene that binds σ^{70} RNA polymerase to globally regulate gene expression.

taxa: γ-proteobacteria only

new homologs: δ-proteobacteria (*Geobacter sulfurreducens*), β-proteobacteria (*Nitrosomonas europaea*, *Chromobacterium violaceum* and *Bordetella*) and *Magnetococcus*, an uncharacterized proteobacteria.

analysis: Hits are confirmed by previous study [6]. γ- and β-protebacteria have diverged ~2 billion years ago [8], so the homologs are highly diverged. (I found no divergence time estimates for δ-proteobacteria.)

RF00042 *cis*-acting antisense RNA, on reverse strand of 5' end of *repA* (plasmid replication) gene

taxa: enterobacteria, which are γ-proteobacteria

new homologs: At E-value 0.05, *Salmonella typhimurium* (not new species). At E-value 0.3, *Aeromonas salmonicida* (diverged γ-proteobacteria, and not enterobacteria) At E-values 0.1-0.7, Spirochaete and a Firmicute (would be extremely diverged)

analysis: New *Salmonella* hit is antisense to start of *repY* gene; *repY* is unexpected (not *repA*), but still replication-related. *Aeromonas* hit is nicely antisense to start of repA gene, so is likely correct. The Spirochaete and Firmicute hits are not on plasmids; they are very suspicious.

RF00080 Putative RNA

taxa: *E. coli* and close enterobacteria relatives

new homologs: *Xanthomonas*, diverged γ-proteobacteria, and not enterobacteria

analysis: A hit was found that intersects the *yybP* RNA motif [5], a conserved RNA found to intersect RF00080 (J.E. Barrick, personal communication). The E-value of the hit was 8.2, which is not very significant. However, there are only 13 hits with a lower

E-value, including RF00080 members, so this could have led to the discovery of the larger *yybP* motif.

RF00185 conserved element in 3' UTR of viral RNA genomes, probably related to replication

taxa: Japanese encephalitis group, a type of flavirus

new homologs: a broader spectrum of flaviruses: Yellow Fever, Dengue Fever, Tick-borne encephalitis

analysis: Hits in 3' UTR of viruses, as expected. Note: based on RSEARCH results of families RF00185 and RF00497, I decided to do additional iterative searches, and discovered that these families are actually part of an RNA conserved in all known flaviviruses.

RF00209 Pestivirus IRES. The internal ribosome entry site (IRES) is an alternate means to initiate translation

taxa: pestiviruses, a type of flaviviridae. (Note: flaviviridae is a broader group that contains flaviviruses.)

new homologs: Hepatitis C, another flaviviridae

analysis: The hit is a known IRES in Hepatitis C. Although this is not a new discovery (since the IRES in Hepatitis C was known through other work), it is nonetheless a significant jump in taxon, and lead to a true homolog.

RF00230 T-box, a regulatory RNA upstream of amino acid biosynthesis operons, regulates the operons based on levels of charged tRNAs of the relevant species

taxa: Firmicutes and one Actinobacteria (*Symbiobacterium*)

new homologs: in δ-proteobacteria (*Geobacter metalireducens* and *Geobacter sulfurreducens*); a different bacterial group that is evolutionarily deeply divergent from Firmicutes and Actinobacteria

analysis: New homologs were upstream of leucine biosynthesis genes. Their "specifier" codons, to which uncharged tRNAs bind, correspond to leucine. (Note: there is a prior claim of a *Geobacter* T-box [136], although no statement of nucleotide coordinates or sequence.)

RF00233 This 3' element resembles a tRNA, and enhances translation in the host

taxa: various pomoviruses, tymoviridae and furoviruses; all are classified as single-stranded positive strand viruses with no DNA stage

new homologs: Other viral groups that are single-stranded positive strand viruses with no DNA stage: tobamovirus (tobacco mosaic virus, Frangipani mosaic virus) and pecluvirus (peanut clump virus). Other species in a known group: pomovirus (beet soil-borne virus, Chinese wheat mosaic virus). Additional homologs in known species.

analysis: I have confirmed all hits in new viral groups (tobamovirus and pecluvirus) are in 3' UTR of genome, as expected.

RF00460 Polyadenylation Inhibition Element: this element is in the 3' UTR of U1A spliceosomal proteins, which bind their own 3' UTR to regulate themselves.

taxa: Mammals

new homologs: zebrafish

analysis: Zebrafish hit is in 3' UTR of U1A gene, as expected. Further work with iterative searches uncovered homologous RNAs in other fish and in frog.

RF00466 The ROSE element is 5' UTR to heat shock genes, and is thought to change its structure in response to temperature

taxa: *Agrobacterium tumefaciens* a member of the *Rhizobiales* group of α-proteobacteria

new homologs: *Gluconobacter oxydans*, a different type of α-proteobacteria (not *Rhizobiales*) and *Rhizobiales* (*Mesorhizobium loti*, *Sinorhizobium meliloti*)

analysis: All hits are on heat shock proteins, as expected. Interestingly, all species are associated with plants: *Rhizobiales* are generally nitrogen-fixing plant symbionts, *Agrobacterium tumefaciens* is a plant parasite and *Gluconobacter oxydans* lives on plants. Some hits of this family intersect RF00435; these two Rfam entries appear to be the same RNA family.

RF00497 This RNA element appears in the 3' UTR of the viral RNA genome, and may play a role in replication

taxa: Dengue virus, a type of flavivirus

new homologs: Several other types of flavivirus: Japanese encephalitis, West Nile virus, Kunjin virus, Murray Valley encephalitis virus and Usutu virus

analysis: The hits span many types of flavivirus, and are all in the 3' UTR of the genome, as expected. Subsequent iterative searches I did found homologs in essentially every sequenced flavivirus that is known to infect a mammal and be carried by mosquito or tick, and discovered that Rfam ID RF00185 is part of the same family.

Families expanded by RSEARCH: moderate expansion and strong evidence

RF00040 Regulatory RNA; bound by RNase E protein to regulate expression

taxa: Enterobacteria (group within γ-proteobacteria)

new homologs: *Photorhabdus luminescens*, a more diverged enterobacteria

analysis: 5' of RNase E gene, as expected. Interestingly, RSEARCH reports a highly significant E-value of 10^{-16}, yet the Rfam CM scores it 15 bits below the threshold.

RF00223 BiP IRES. The internal ribosome entry site (IRES) is an alternate means to initiate translation; this one is specific to BiP, a heat shock gene.

taxa: humans only

new homologs: mice

analysis: At start of translation site in heat shock gene—exactly as expected. Note: the mouse sequence is a partial sequence, so Rfam's CM may have difficulty with the truncated RNA.

RF00224 FGF2 IRES. The internal ribosome entry site (IRES) is an alternate means to initiate translation; this one is specific to fibroblast growth factor 2 (FGF2) genes.

taxa: human only

new homologs: cow

analysis: At start of translation site in FSF2 gene—exactly as expected. Note: the cow sequence is a partial sequence, so Rfam's CM may have difficulty with the truncated RNA.

RF00252 RNA in 3' UTR of viral coat protein that is bound by the protein

taxa: alfalfa mosaic virus, a kind of alfamovirus, which is in turn a bromoviridae, a type of single-stranded positive strand RNA virus

new homologs: Humulus japonicus latent virus, an Ilarvirus, which is a bromoviridae; also a marginal E-value (0.66) Maize stripe virus, a type of negative-strand RNA virus

analysis: Humulus japonicus latent virus is 3' of coat protein, as expected. (The Maize stripe virus is 3' of a protein, but the protein is "non-capsid protein". This is probably a false positive, but may be a different 3' UTR viral element.)

RF00434 element in 3' UTR of viral RNA genome mediates RNA translation

taxa: luteoviridae and one umbravirus

new homologs: Groundnut rosette virus, a new umbravirus, plus a new hit in Barley yellow dwarf virus, an already-known species

analysis: Both hits are in 3' UTR of phylogenetically plausible species. The Barley yellow dwarf virus hit is in a partial sequence; Rfam's CM may have had trouble with the truncated RNA.

RF00435 The ROSE element is 5' UTR to heat shock genes, and is thought to change its structure in response to temperature

taxa: *Bradyrhizobium japonicum* a member of the *Rhizobiales* group of α-proteobacteria

new homologs: *Mesorhizobium loti*, also *Rhizobiales*

analysis: Hits are upstream of heat shock genes, as expected. Some hits with higher E-values in more diverged *Rhizobiales* intersect with RF00466, another ROSE element. (This was noticed based on results from RF00466.)

RF00470 5' plus strand element necessary for RNA synthesis

taxa: alphaviruses

new homologs: Highlands J virus, an alphavirus

analysis: In expected location. The new hit is in a partial sequence; Rfam's CM may have difficulty with the truncated RNA.

RF00500 This element in the 3' UTR of the viral RNA genome can repress synthesis of a complementary strand of the single-stranded RNA virus and thereby affect replication

taxa: turnip crinkle virus, a carmovirus

new homologs: Cardamine chlorotic fleck carmovirus, also a carmovirus

analysis: Hit is in 3' UTR of virus, as expected.

Families expanded by RSEARCH: modest expansion and strong evidence

RF00048 *cis*-acting viral replication element, found within coding frame of protein 2C

taxa: enteroviruses

new homologs: bovine enterovirus (already known)

analysis: Within coding region of protein 2C, as expected, although E-value is marginal (0.5).

RF00061 Hepatitis C internal ribosome entry site, a regulatory RNA structure

taxa: Hepatitis C

new homologs: 2325 hits within Hepatitis C and Hepatitis GB

analysis: Inspection of a few revealed they were close to translation start, as expected.

RF00165 pseudoknot in 3' UTR of viral RNA genome

taxa: coronaviruses

new homologs: human coronavirus

analysis: In 3' UTR, as expected.

RF00168 Lysine-binding riboswitch

taxa: γ-proteobacteria

new homologs: *Haemophilus ducreyi*, related to the *H. influenzae* already known

analysis: Upstream of *lysC*, as expected.

RF00171 element in 5' UTR of viral RNA; replication defect phenotype

taxa: Tombusviridae

new homologs: Johnsongrass chlorotic stripe mosaic virus, a type of Tombusviridae

analysis: In 5' UTR, as expected.

RF00175 element in 5' of genome involved in viral packaging and replication

taxa: HIV, SIV (infects other primates) and CIV (goats)

new homologs: HIV and SIV

analysis: There are many hits; checks of about one quarter of them indicate they are in the 5' part of the genome, as expected. Many are in partial sequences, where Rfam's CM may have had trouble recognizing a chopped version of the full RNA element.

RF00194 3' element to protein E in rubella virus genomes

taxa: rubella

new homologs: rubella

analysis: Several new hits, all 3' to protein E, as expected.

RF00210 Aphthovirus IRES. The internal ribosome entry site (IRES) is an alternate means to initiate translation

taxa: some aphthoviruses (e.g., foot & mouth disease)

new homologs: a broad variety of picornaviruses, of which aphthoviruses are one type, and one herpesvirus (cytomegalovirus).

analysis: There is significant evidence to support virtually all of the hits (some

overlap annotated IRESes, and most are close to beginning of start of translation of a gene, as expected). Unfortunately, BLAST queries indicate that the sequence may be used in a cloning vector. Therefore, an unknown number of these hits are probably sequence contamination. However, they are, at least, homologs that Rfam's CM does not detect.

RF00236 *cis*-acting antisense RNA, on reverse strand of 5' end of *repA* (plasmid replication) gene

taxa: *Corynebacterium*

new homologs: *Corynebacterium efficiens* (known species)

analysis: Overlaps 5' end of uncharacterized gene in plasmid. Based on BLAST searches, this gene has homology to repA genes in other plasmids, and the RF00236 hit overlaps its predicted Shine-Dalgarno sequence (J.E. Barrick, personal communication).

RF00468 RNA in the viral NS5B coding region, exact function unknown

taxa: Hepatitis C

new homologs: also Hepatitis C

analysis: Hits in NS5B coding region, as expected. All new hits are in partial sequences; Rfam's CM may have difficulty with the truncated RNA.

RF00498 This element is in the 5' UTR of viral RNA genomes, and is used to transcribe shorter mRNAs from the viral RNA

taxa: arterivirus

new homologs: known species

analysis: Hits are in 5' UTR, as expected.

RF00501 This RNA element is found in the 3' end of rotavirus mRNAs

taxa: rotaviruses

new homologs: a known-species of rotavirus

analysis: In 3' UTR, as expected. All new hits appear to be in partial sequences; Rfam's CM may have trouble with the truncated RNAs.

Families expanded by RSEARCH: significant expansion and plausible/unclear evidence

RF00021 Spot 42, a *trans*-acting antisense RNA gene

taxa: γ-proteobacteria

new homologs: *Pseudomonas syringae*, a plausible but diverged extension of the family

analysis: No evidence either way. However, E-value is marginal (0.5).

RF00101 putative RNA

taxa: enterobacteria

new homologs: *Photorhabdus*, a diverged enterobacteria with strong E-value; more diverged *Shewanella oneidensis* (also a γ-proteobacteria, but not an enterobacteria) at marginal E-value 0.1

analysis: No evidence for or against.

RF00111 putative RNA

taxa: enterobacteria

new homologs: *Photorhabdus*, a diverged enterobacteria and more diverged *Shewanella oneidensis* (also a γ-proteobacteria, but not an enterobacteria).

analysis: No evidence for or against. (Also finds an α-proteobacteria and *Vibrio*, a more diverged γ-proteobacteria, with marginal E-values, but these overlap predicted genes.)

RF00112 putative RNA

taxa: enterobacteria

new homologs: *Photorhabdus* and *Erwinia*, diverged enterobacteria

analysis: Little evidence for or against. Hits are adjacent to *yegQ*, like the known hits. However, they are not near *orgK* as the previously known hits were. It is possible that the proximity to *orgK* was spurious—indeed both genes' proximity may be an accident of phylogenetic correlation.

Families expanded by RSEARCH: significant expansion and dubious/weak evidence

RF00035 *trans*-acting antisense RNA

taxa: *E. coli* & closely related γ-proteobacteria

new homologs: cyanobacteria

analysis: Homology is subjectively not compelling; overlaps 5' end of hypothetical but conserved protein. At 0.0008, this is the lowest E-value I found where the hit was dubious at best.

RF00115 putative RNA

taxa: enterobacteria

new homologs: *Desulfotalea psychrophila*, an intimidating-sounding δ-proteobacteria

analysis: No strong evidence for or against, but jump in taxon is suspicious, and E-value is marginal (0.11)

RF00116 putative RNA

taxa: *E. coli* and *Shigella* only

new homologs: *Yersinia*, a highly diverged enterobacteria when compared to the two restricted known enterobacteria.

analysis: Little evidence before or against. The hit is not between the genes that the known hits are between. However, being between genes in such closely related species could easily be a coincidence.

RF00166 Possible CsrA protein-binding RNA

taxa: *Pseudomonas*

new homologs: *Azotobacter*, a bit diverged from *Pseudomonas*; and *Bacteroides thetaiotaomicron* is a Bacteriodes group bacteria, and apparently this species has joined a college fraternity.

analysis: *Azotobacter* hit is probably correct, based on phylogeny. *Bacteroides* hit is suspicious phylogenetically, but homology is subjectively fairly good throughout the RNA,

despite the presence of a possible transcription terminator loop, which is a confounding factor.

RF00238 *cis*-acting antisense RNA, on reverse strand of 5' end of *repA* (plasmid replication) gene

taxa: *Lactococcus*, a Firmicute

new homologs: *Borrelia burgdorferi*, in Spirochaete, a different bacterial group

analysis: Unclear. Taxon jump is suspicious, but it is reassuring that at least the hit is in a plasmid. There is no annotated gene in the immediate vicinity of the putative homologs; it is possible that an unrecognized replication gene is here. The plasmid has no annotate *rep* gene, although the annotation has few genes annotated as anything other than "hypothetical". The hit's E-value is marginal (0.3).

RF00242 *cis*-acting antisense RNA, on reverse strand of 5' end of *repA* (plasmid replication) gene

taxa: *Staphylococcus*, a Firmicute

new homologs: *Mycoplasma mycoides*, also Firmicutes

analysis: Doubtful. The two species are not too distantly related, but suspiciously, (1) the hit is not in a plasmid, (2) the hit overlaps a predicted tRNA; unrelated RNAs are a common source of false positives in homology searches.

RF00370 putative RNA

taxa: *E. coli* and close relatives

new homologs: *Ralstonia eutropha*, a β-proteobacteria, a very different group from *E. coli*

analysis: The hit overlaps a predicted gene, which makes it very suspicious, in addition to the extremely diverged species.

RF00433 This *cis*-regulatory RNA is 5' to certain heat shock genes

taxa: fruit fly

new homologs: zebrafish and mouse

analysis: Little evidence to confirm or deny, but somewhat suspicious. E-values are marginal (~0.1). Both hits are close to the 3' end of a gene, making them look like 3' UTR elements, if anything; by contrast it is 5' to the gene in fly. Both hits are in mRNA sequences (in the RefSeq transcript database), and in both cases, BLAST searches using the protein were unable to find a homolog in a larger, well-annotated sequence; consequently, it remains a possibility that one or both hits are 5' to some other gene that is heat shock related.

RF00489 *cis*-acting antisense RNA, on reverse strand of 5' end of *repC* (plasmid replication) gene

taxa: *Rhizobiales*, a sub-type of α-proteobacteria

new homologs: *Pseudomonas*, a γ-proteobacteria with E-value 0.07

analysis: Doubtful. It is between genes that do not appear replication-related, and moreover it is 3' relative to both genes, where it should be on the 5' end of at least one. In view of this and the surprising phylogenetic jump, this hit is very dubious.

RF00506 This RNA is found in the 5' UTR of bacterial threonine synthesis operons, and regulates gene expression

taxa: enterobacteria

new homologs: wide variety of bacterial groups

analysis: None of the new hits were 5' to a threonine operon, or indeed any gene. Moreover, most matches were to the transcription termination loop part of the RNA; this is likely a transcription terminator loop used for another purpose, such as terminating transcription in the normal way, 3' to a coding region. Transcription terminator loops are a confounding factor in homology searches, particularly with local CMs that permit matching only the loop.

Families expanded by RSEARCH: other

RF00034 *trans*-acting antisense ncRNA gene

taxa: enterobacteria

new homologs: *Photorhabdus*, a diverged enbacteria

analysis: Plausible, but no evidence to support or refute.

RF00064 putative RNA

taxa: various species of *Pyrococcus*

new homologs: *Pyrococcus*

analysis: No evidence either way.

RF00081 putative RNA

taxa: enterobacteria

new homologs: *Photorhabdus* and *Erwinia*, two diverged enterobacteria at E-value $\sim 10^{-5}$; α-proteobacteria and cyanobacteria at E-values 0.3 and 0.5.

analysis: No evidence to support or deny, but new enterobacteria seem plausible. The α-proteobacteria and cyanobacteria are suspicious.

RF00083 putative RNA

taxa: enterobacteria

new homologs: *Photorhabdus*, *Erwinia* and *Yersinia*, diverged enterobacteria

analysis: E-values are very significant, but there was no evidence for or against. Also a more diverged *Vibrio* hit with marginal E-value 0.3.

RF00121 putative RNA; may regulate other genes

taxa: *E. coli* and close enterobacteria relatives

new homologs: *E. coli* at E-value 0.01 and 0.03.

analysis: Possible. One hairpin looks like a possible transcription terminator loop, which is a common source of false positives in RNA homology searches. However, this RNA

is not known to be *cis*-regulatory. A hit in more diverged γ-proteobacteria *Shewanella* at E-value 8 has subjectively better homology.

RF00124 putative RNA

taxa: *E. coli* and *Shigella*

new homologs: *E. coli* O157:H7, an *E. coli* strain not in the known members.

analysis: Upstream of same genes as known RNAs. This argues for homology, since the most likely explanation is that the genes (in *E. coli* O157:H7 and other strains) are homologous, and that the nearby putative RNA is also homologous. (This explanation seems most parsimonious even if the RNA is independently transcribed, and has no functional relationship to the gene.) More information may be available if the function of the RNA were known.

RF00127 putative RNA

taxa: γ-proteobacteria

new homologs: New hits in more diverged γ-proteobacteria *Microbulbifer degredans* and *Pseudomonas fluorescens.*

analysis: No evidence for or against. There are many hits with less significant E-values that are also within γ-proteobacteria; given an automated RNA motif discovery tool, it may be worth trying to see if these other hits can be incorporated into a new alignment.

RF00128 putative RNA

taxa: enterobacteria

new homologs: *Erwinia*, a diverged enterobacteria

analysis: No evidence for or against.

RF00193 viral replication signal RNA in viral 3' UTR

taxa: citrus virus, a closetrovirus

new homologs: beet yellow virus, also closetrovirus

analysis: Plausible genome location (almost in 3' UTR) and species, but overlaps

annotated protein-coding gene by ~150 nucleotides. Although annotation of protein-coding genes is generally more reliable than ncRNAs, it is striking that the species is plausible and the homolog is roughly in the 3' UTR.

RF00229 Picornavirus IRES. The internal ribosome entry site (IRES) is an alternate means to initiate translation

taxa: picornaviruses

new homologs: no new species, but 429 new hits

analysis: All hits are plausible, but the partial sequences and incomplete annotation make it difficult to evaluate. The fact that all hits with significant E-values are in species already known to have homologs is encouraging. All 429 new hits appear to be in partial sequences; Rfam's CM may have trouble with the truncated RNAs.

RF00370 putative RNA

taxa: enterobacteria

new homologs: *Yersinia*, slightly diverged enterobacteria

analysis: RNA is hypothetical, so difficult to tell. *Yersinia* is a plausible, slightly diverged species.

RF00444 PrrF is an ncRNA gene that down-regulates several genes

taxa: *Pseudomonas*, a genus of γ-proteobacteria

new homologs: *Azotobacter*, a close genus to *Pseudomonas*; also a hit with E-value of 0.01 in *Deinococcus*, a very distantly related Firmicute

analysis: It is difficult to evaluate the hits, since there are no annotations to support or refute the hit. The *Deinococcus* hit is extremely suspicious based on the genetic divergence, and the fact that no other intervening species were found.

RF00467 RNA element required for initiation of reverse transcription

taxa: Retroviridae and some vertebrates (presumably from retroviral integration events)

new homologs: several already-known types of viruses

analysis: All new hits are in plausible viral species, which is unlikely to occur if they are random false positives. In the Rous sarcoma virus, the RNA is in the 5' UTR in the U5 region. However, other viruses appear to have U5 regions in other parts of their genome. (Some existing Rfam members are, for example, in the 3' UTR of the virus, near a U5 region.) However, some U5 regions are probably not annotated. Therefore, it is likely that the hits are real.

Families not expanded by RSEARCH

The following Rfam IDs are families for which RSEARCH found no additional members. One class of RNAs that was usually unsuccessful were small regulatory RNAs in eukaryotes, which required searching the 540-Mbase RefSeq transcript database; thus, signal-to-noise is poor.

RF00014, RF00022, RF00025, RF00033, RF00034, RF00038, RF00039, RF00043, RF00057, RF00058, RF00060, RF00062, RF00063, RF00077, RF00078, RF00079, RF00080, RF00082, RF00094, RF00102, RF00110, RF00113, RF00117, RF00118, RF00119, RF00120, RF00122, RF00125, RF00126, RF00164, RF00173, RF00176, RF00182, RF00184, RF00195, RF00196, RF00197, RF00215, RF00220, RF00225, RF00227, RF00235, RF00243, RF00250, RF00260, RF00290, RF00362, RF00368, RF00369, RF00371, RF00372, RF00382, RF00383, RF00384, RF00385, RF00386, RF00387, RF00388, RF00389, RF00390, RF00391, RF00436, RF00437, RF00442, RF00447, RF00448, RF00449, RF00453, RF00454, RF00457, RF00458, RF00459, RF00461, RF00462, RF00463, RF00465, RF00469, RF00480, RF00481, RF00483, RF00484, RF00485, RF00487, RF00495, RF00496, RF00499, RF00502, RF00504, RF00505, RF00507.

There are a total of 90 such families.

8.4.2 ROC-like curves

The complete set of ROC-like curves appear on the following pages. Legends:

- BLAST seed 1-sided = BLAST heuristic, but searching only the forward strand; see

section 7.5.6.

- ML-heur = ML-heuristic.
- Rigorous HMM = local rigorous profile HMM of chapter 6.
- NNffCM = discriminative heuristic; see chapter 9.
- NNff1 = discriminative heuristic; see chapter 9.
- NSheur (G+C) = discriminative heuristic; see chapter 9.
- NSheur (E. coli) = discriminative heuristic; see chapter 9.

BLAST performs poorly—especially so for the RSEARCH queries. The ML-heuristic does well, but often the rigorous HMMs do better. The discriminative HMMs generally do well on this data set.

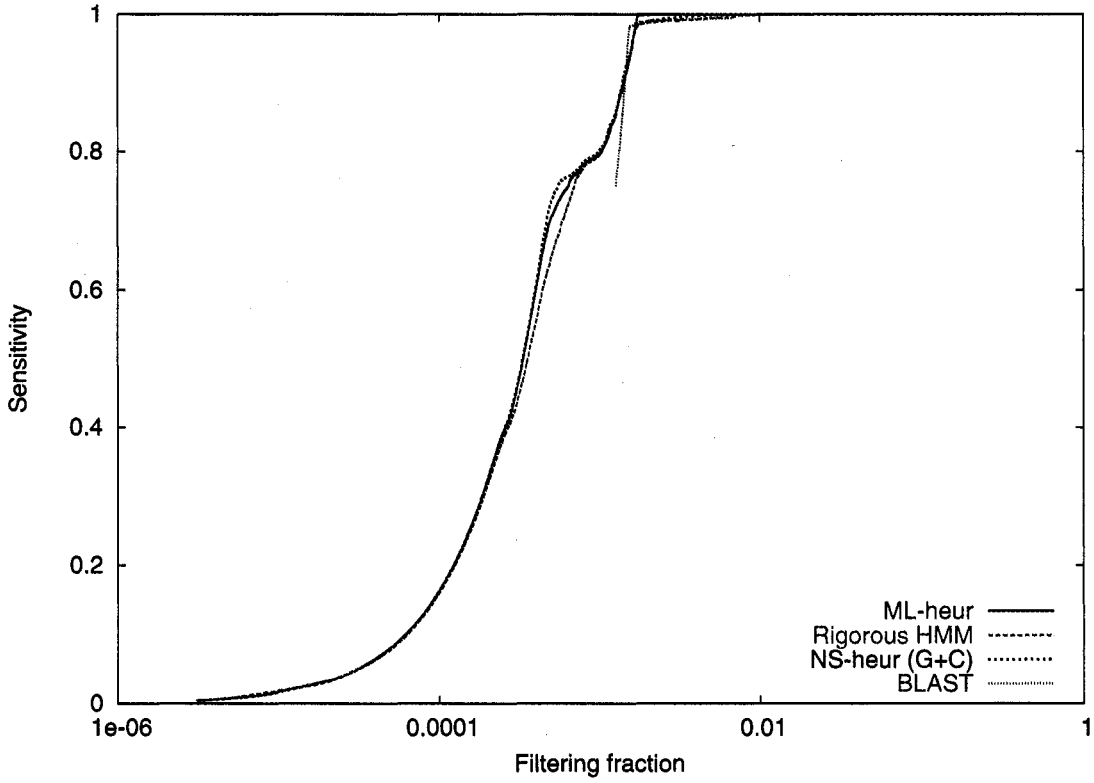


Figure 8.1: ROC-like curve: RF00002

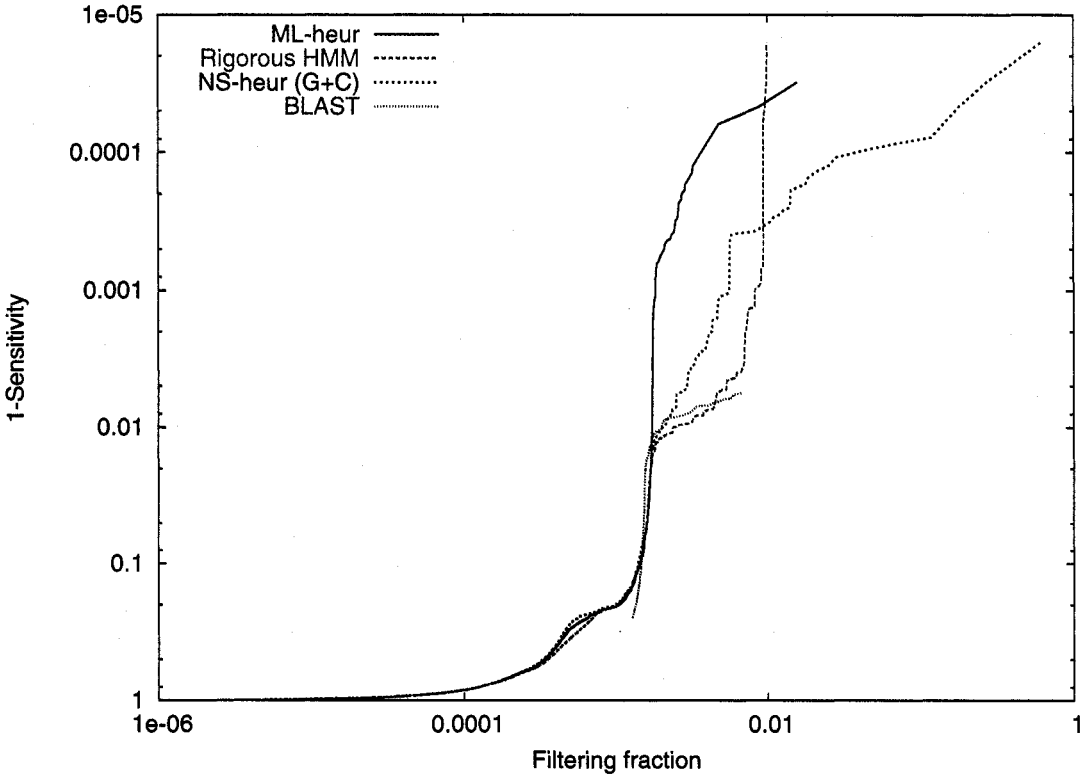


Figure 8.2: ROC-like curve: RF00002 (log-scale sensitivity)

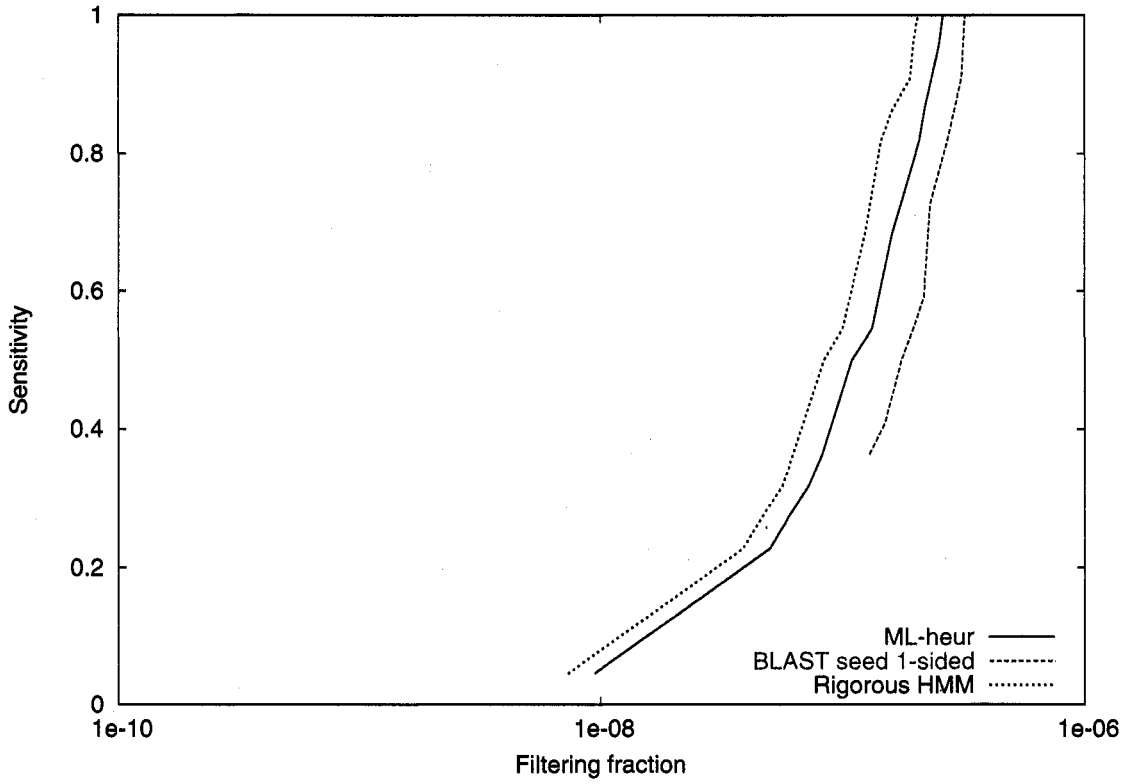


Figure 8.3: ROC-like curve: RF00006

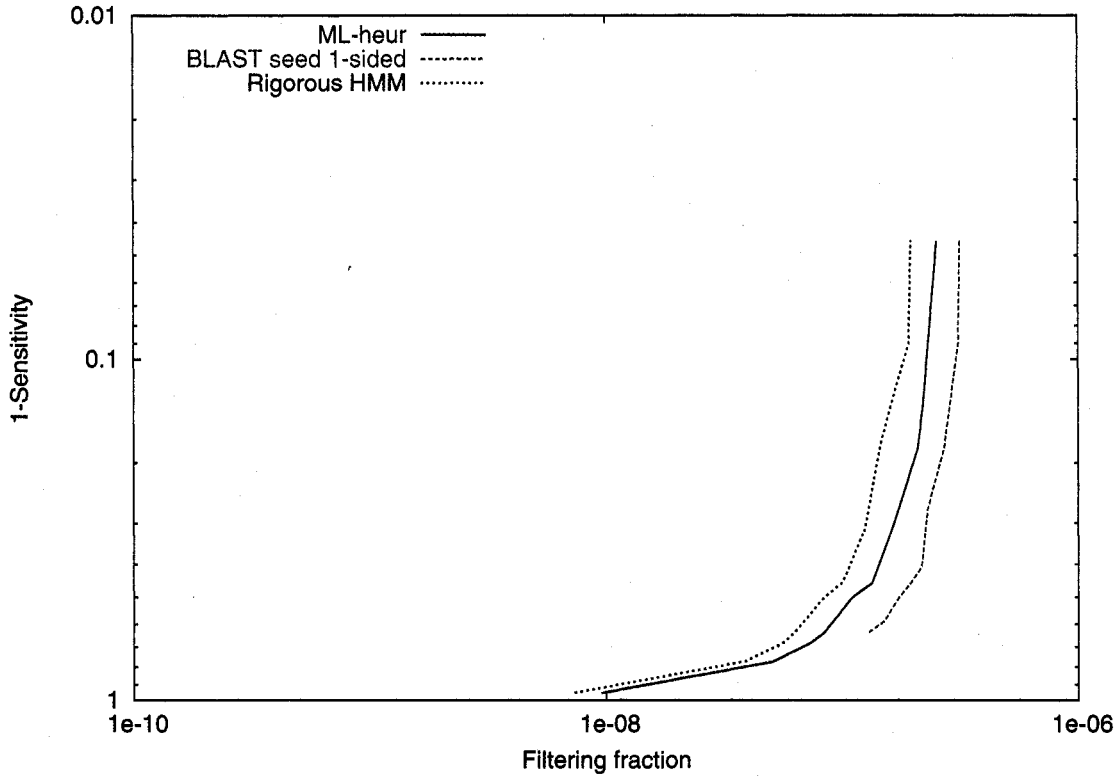


Figure 8.4: ROC-like curve: RF00006 (log-scale sensitivity)

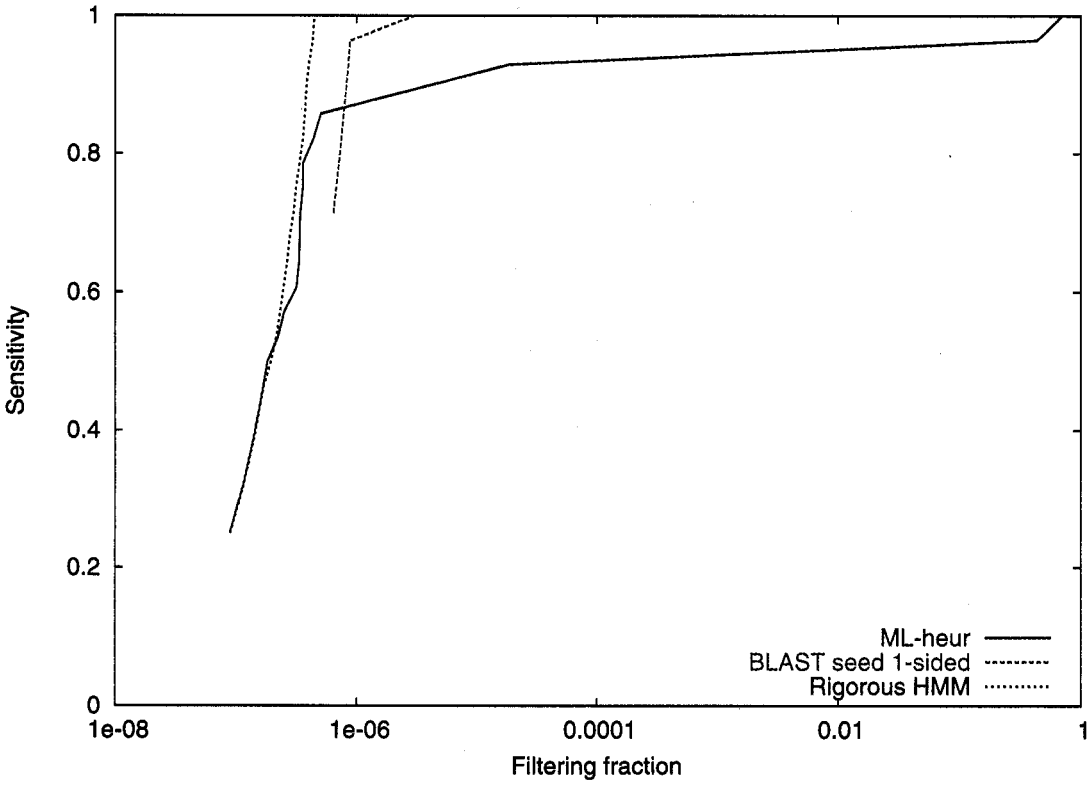


Figure 8.5: ROC-like curve: RF00013

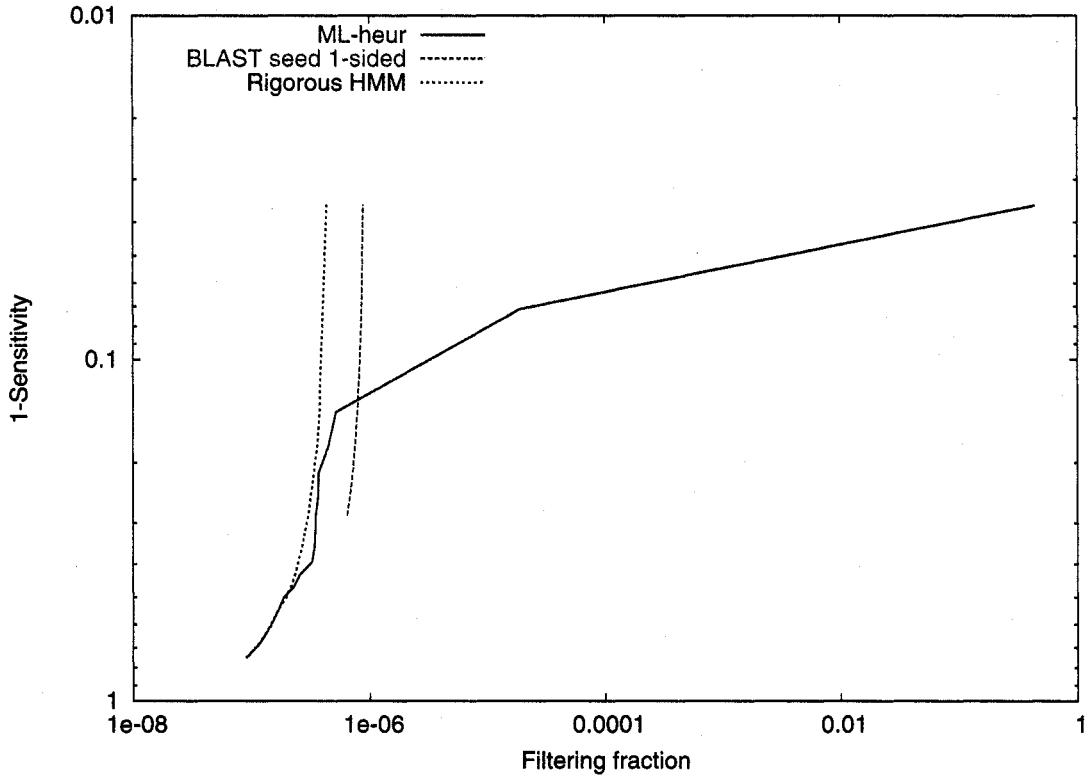


Figure 8.6: ROC-like curve: RF00013 (log-scale sensitivity)

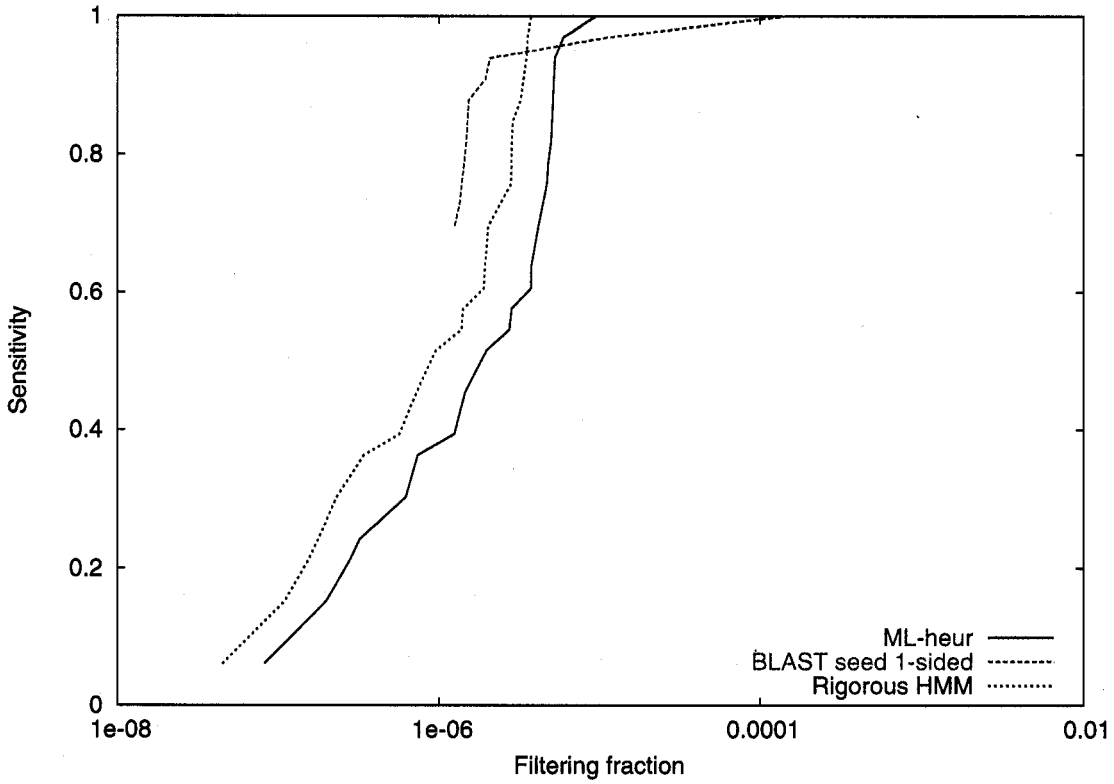


Figure 8.7: ROC-like curve: RF00018

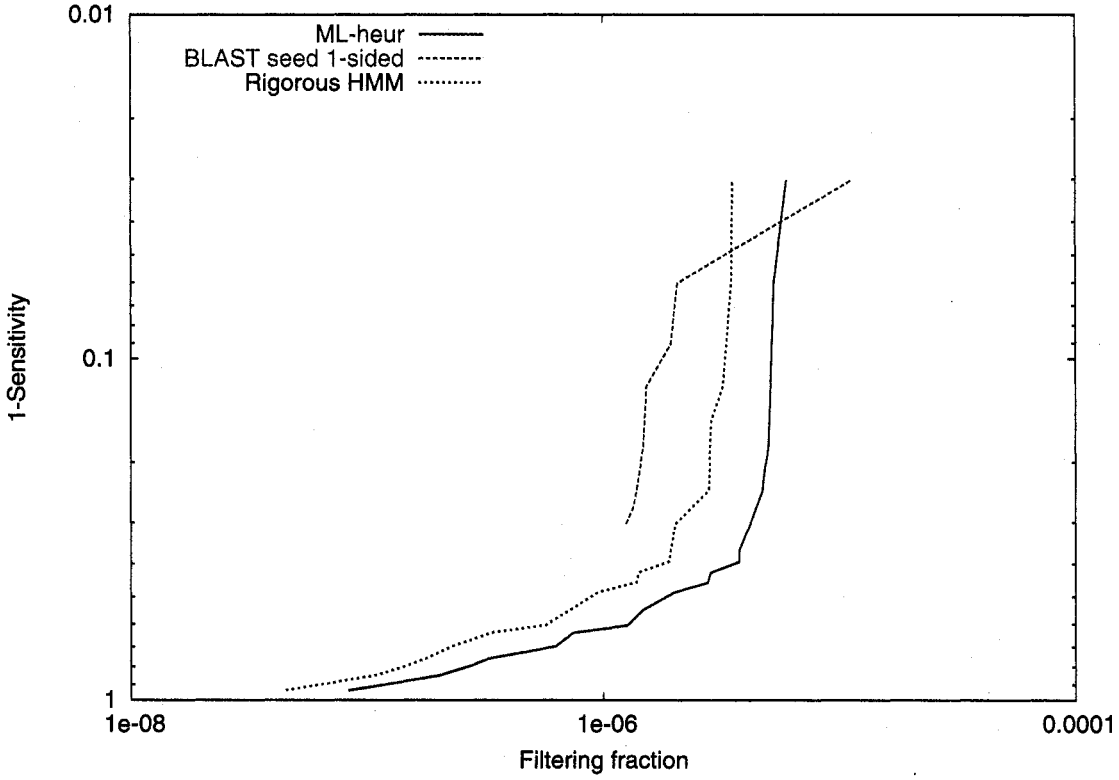


Figure 8.8: ROC-like curve: RF00018 (log-scale sensitivity)

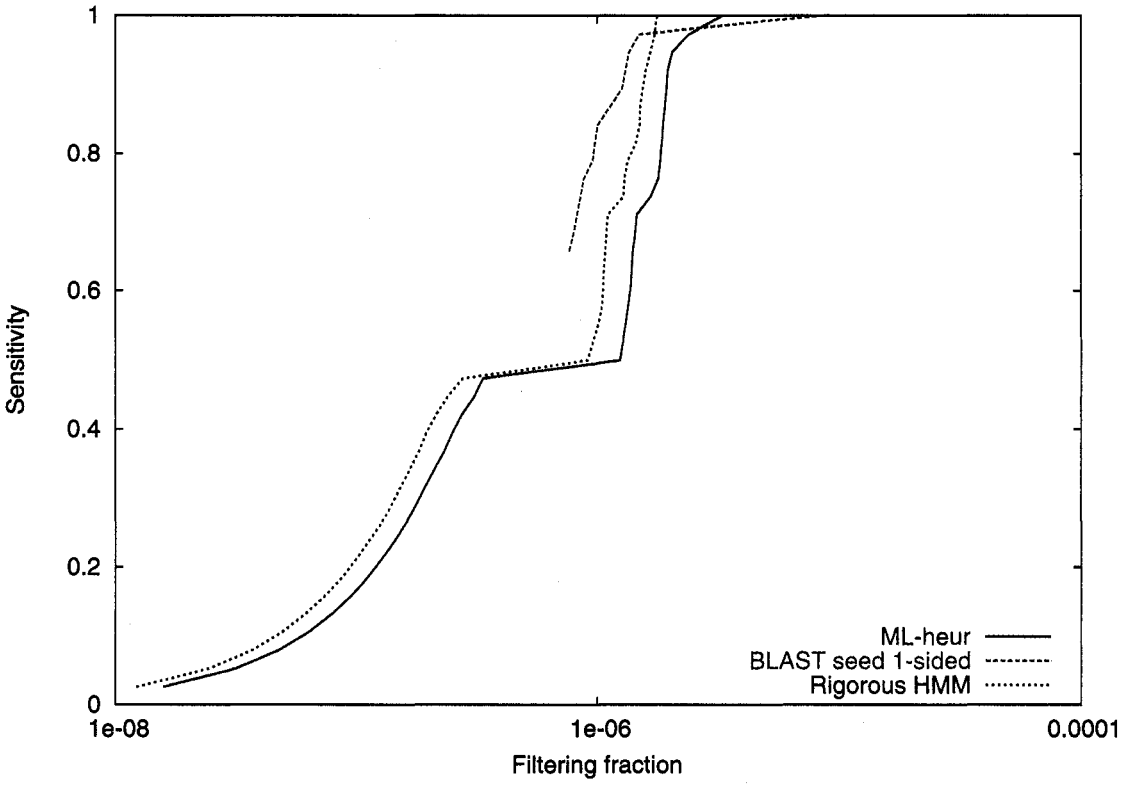


Figure 8.9: ROC-like curve: RF00062

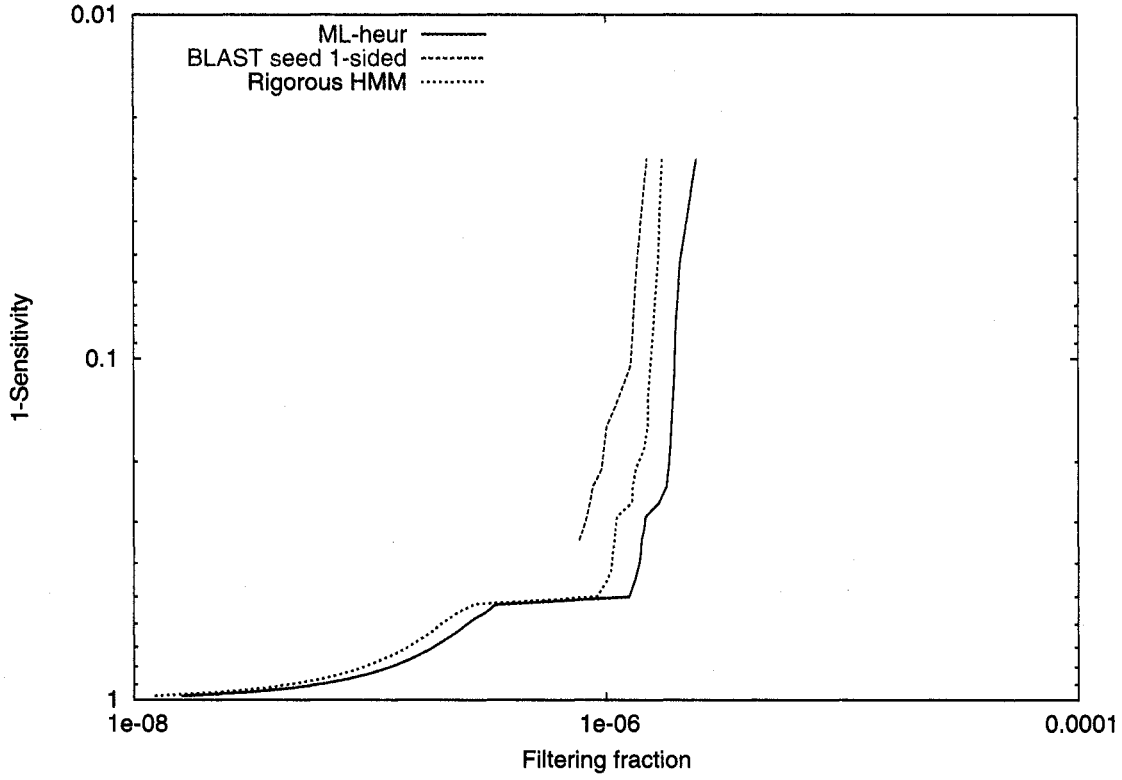


Figure 8.10: ROC-like curve: RF00062 (log-scale sensitivity)

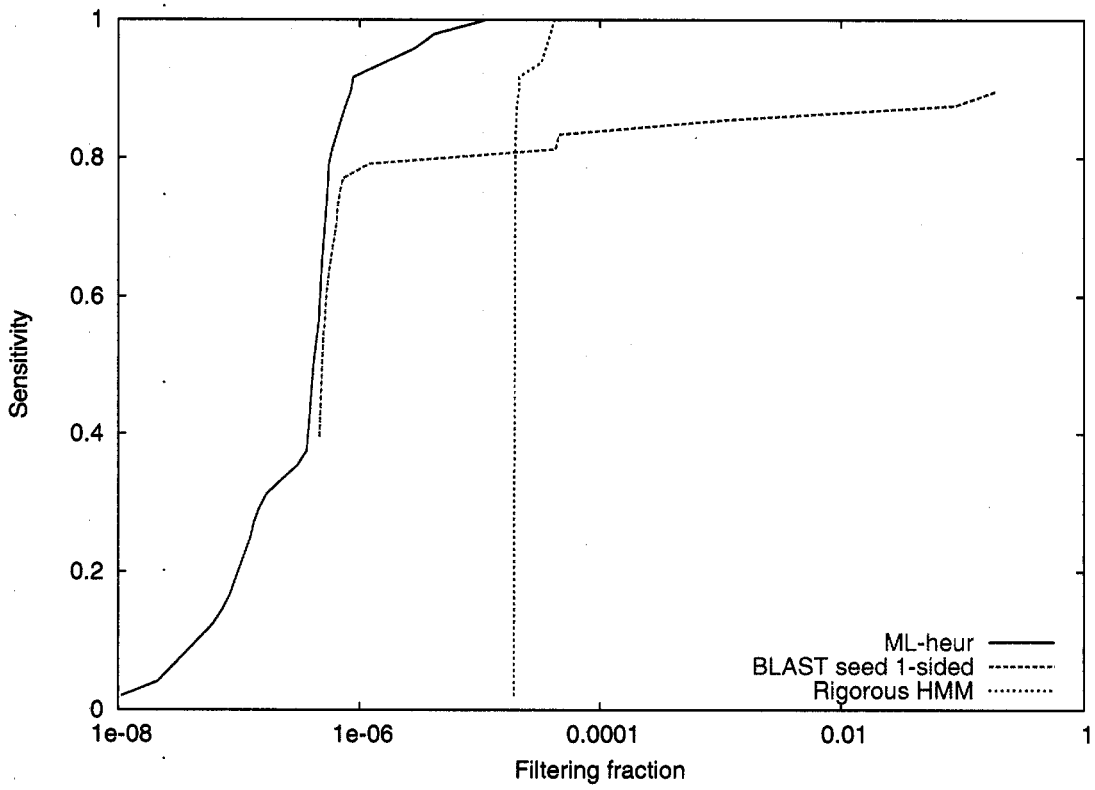


Figure 8.11: ROC-like curve: RF00067

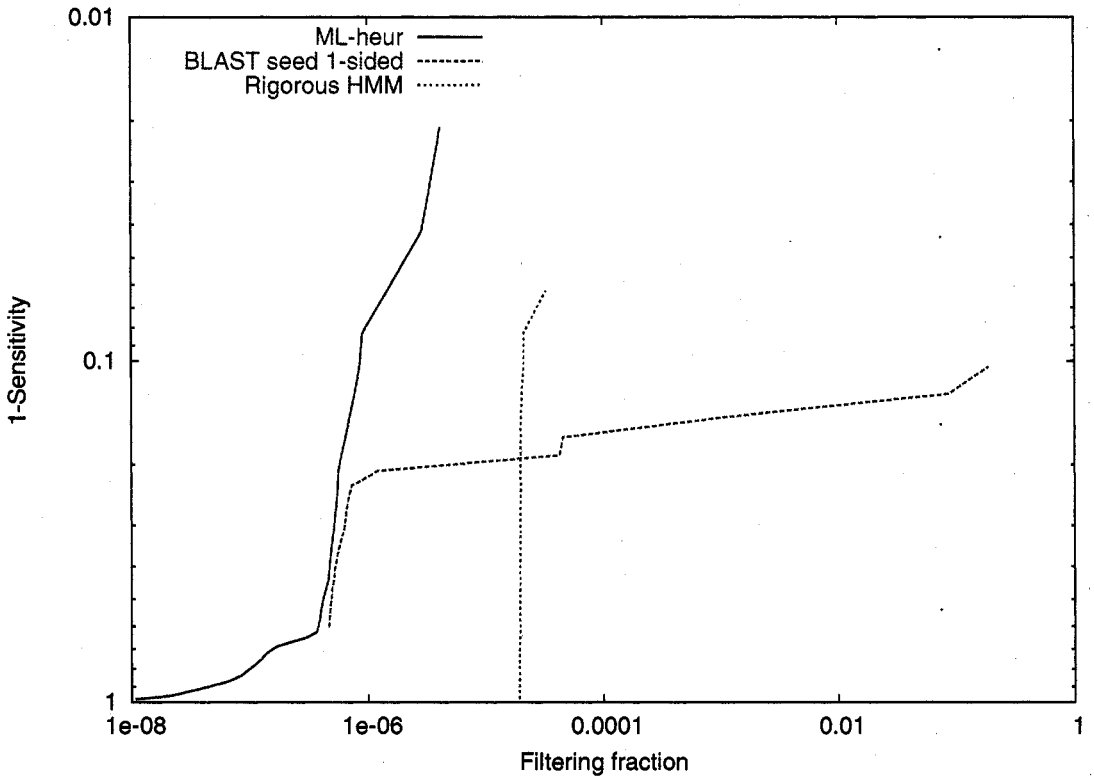


Figure 8.12: ROC-like curve: RF00067 (log-scale sensitivity)

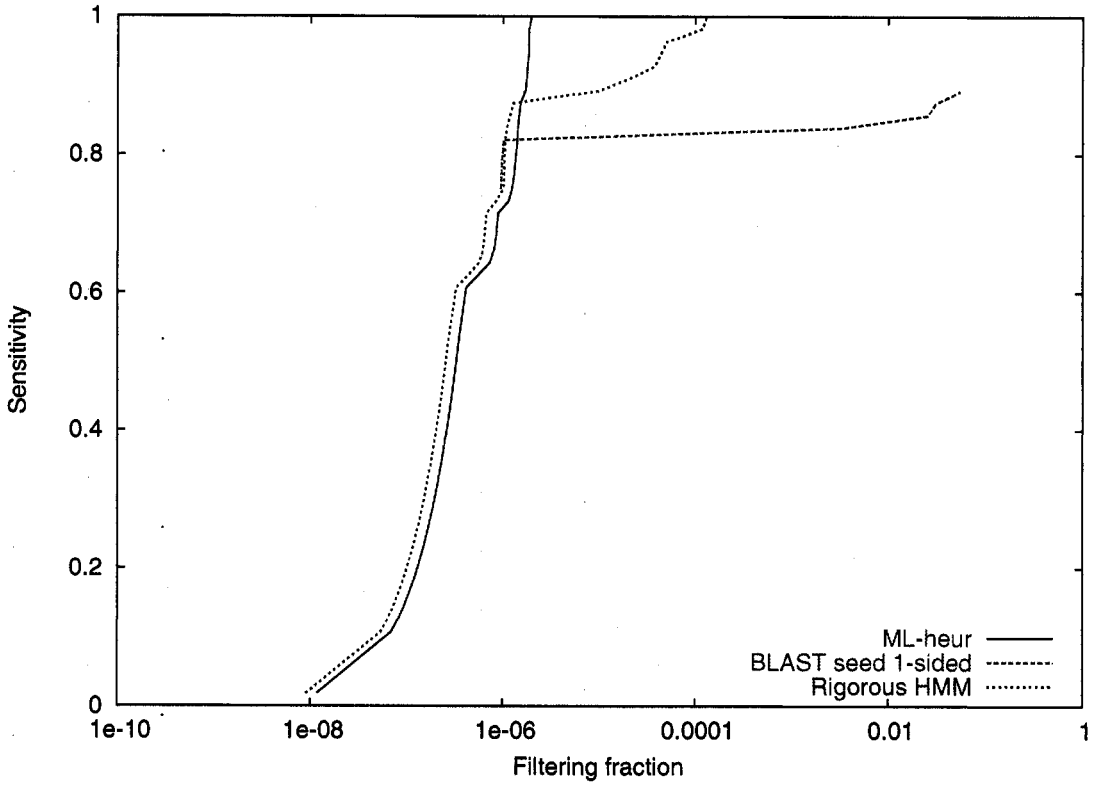


Figure 8.13: ROC-like curve: RF00113

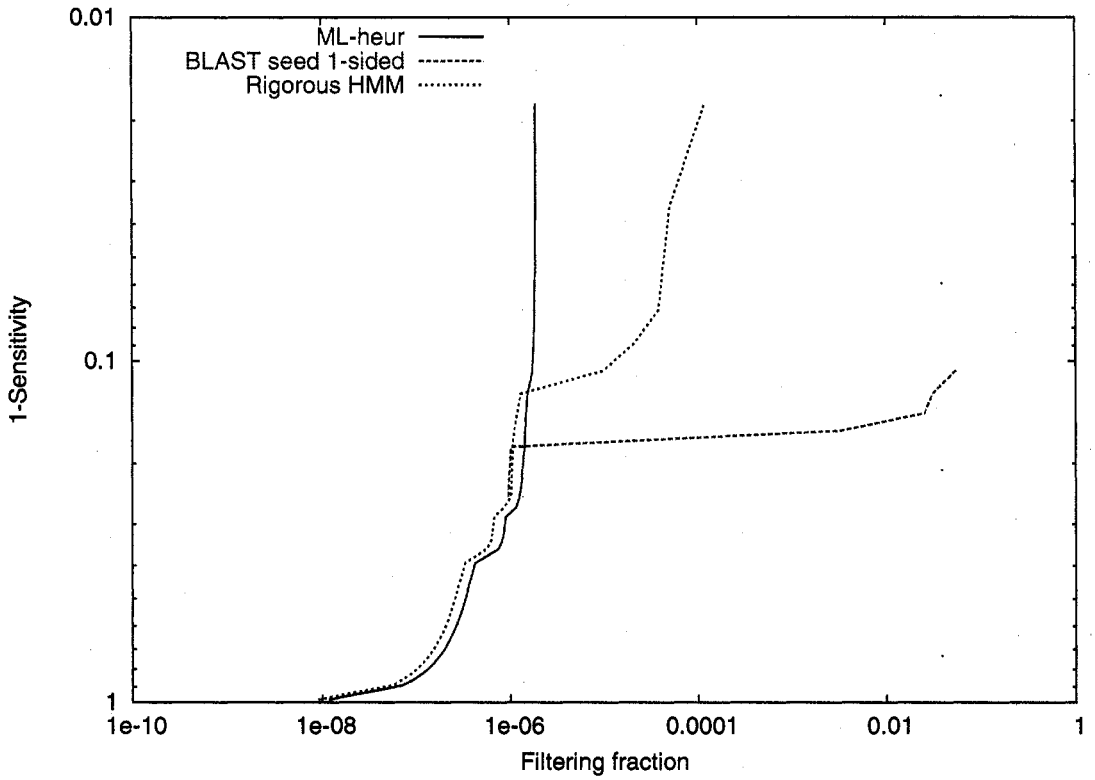


Figure 8.14: ROC-like curve: RF00113 (log-scale sensitivity)

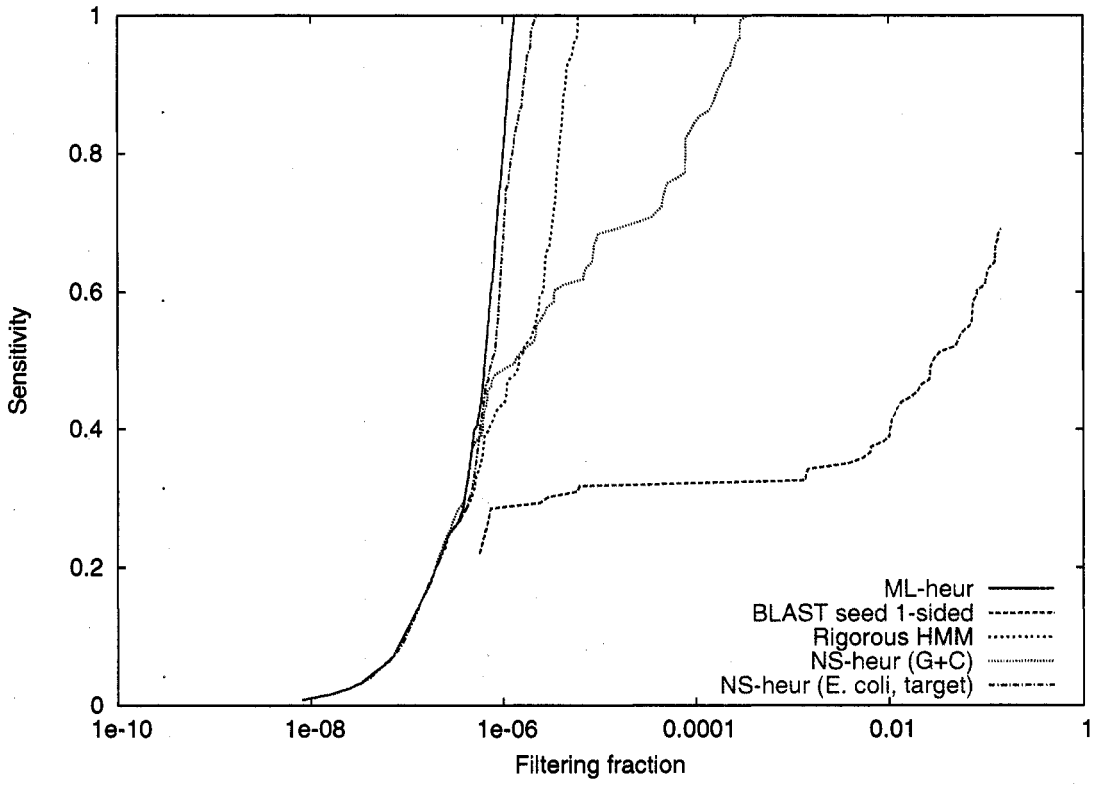


Figure 8.15: ROC-like curve: RF00114

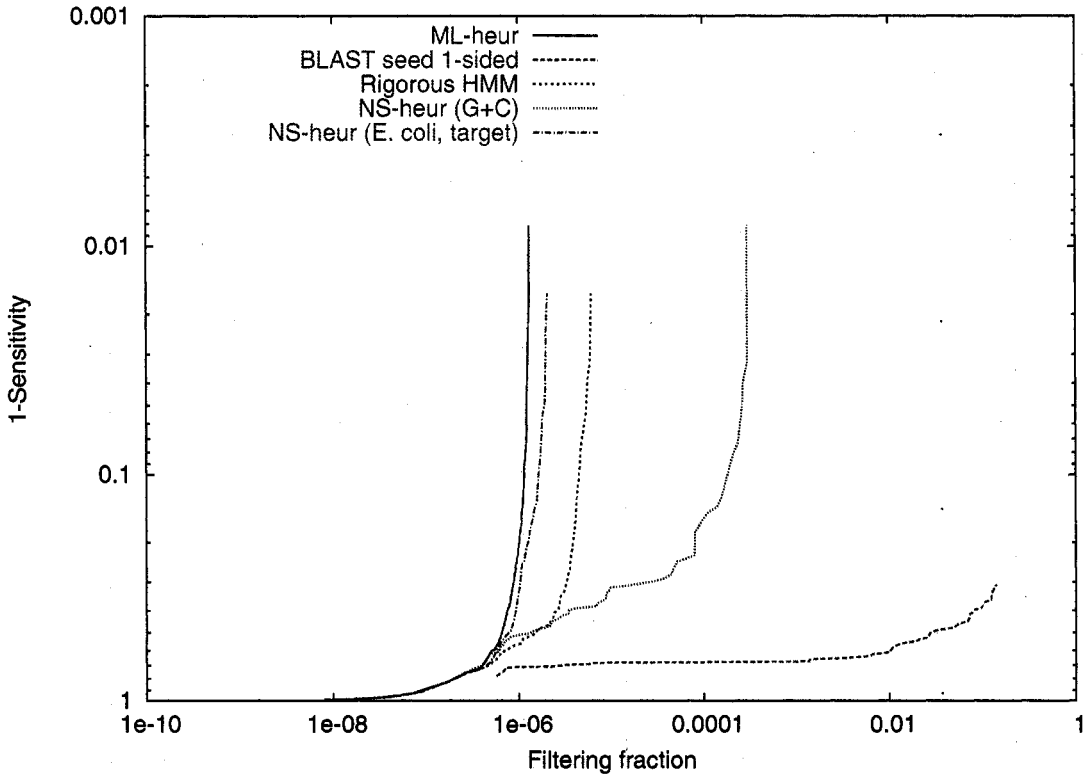


Figure 8.16: ROC-like curve: RF00114 (log-scale sensitivity)

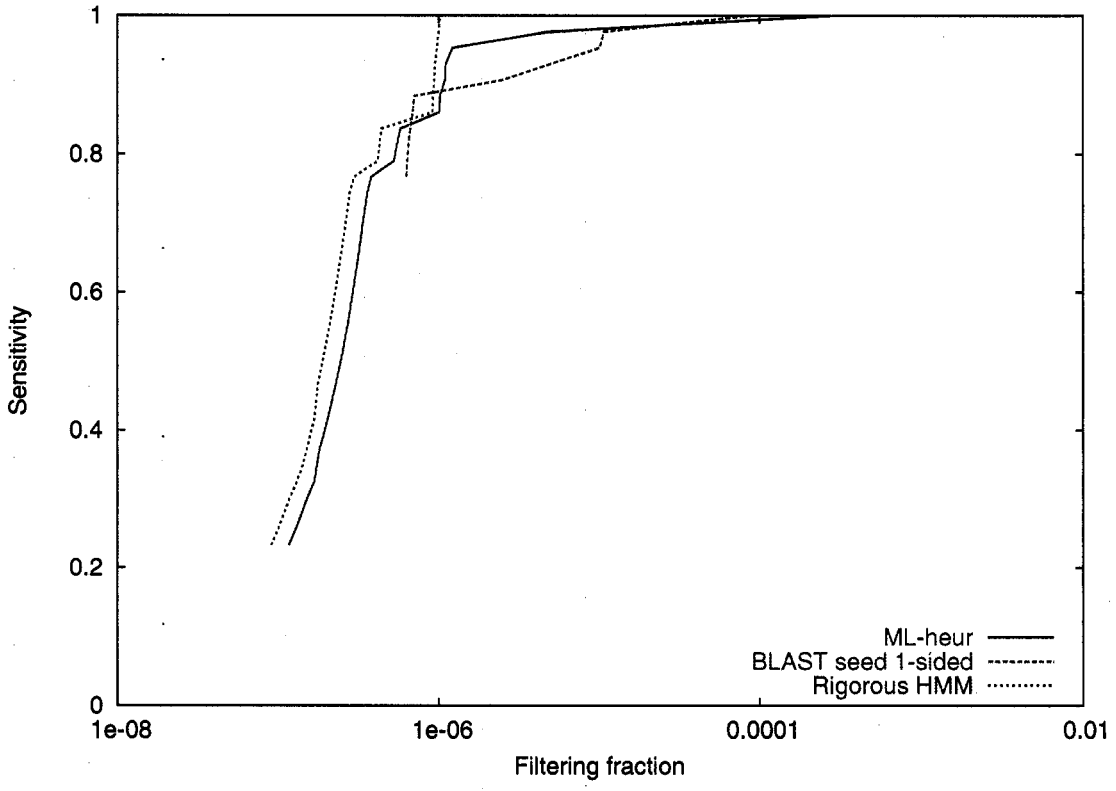


Figure 8.17: ROC-like curve: RF00171

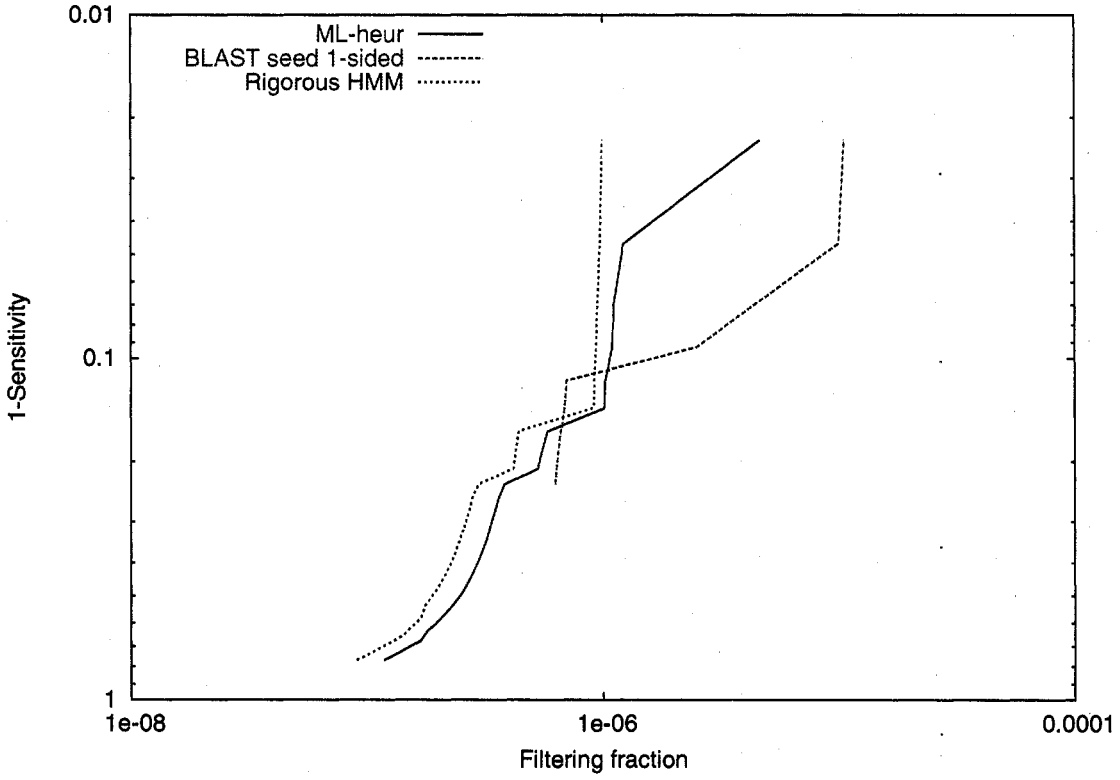


Figure 8.18: ROC-like curve: RF00171 (log-scale sensitivity)

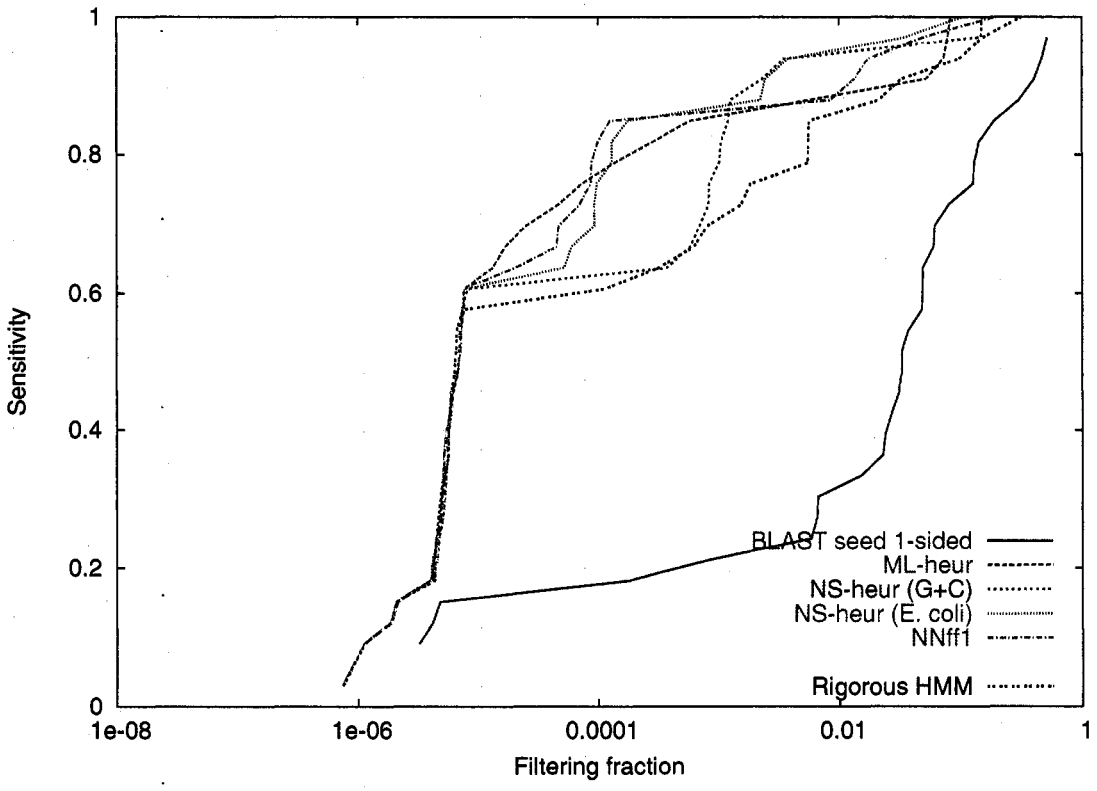


Figure 8.19: ROC-like curve: 6S-Ecoli (RSEARCH; 6S from Rfam family RF00013, on 1.2-gigabase microbe genome database)

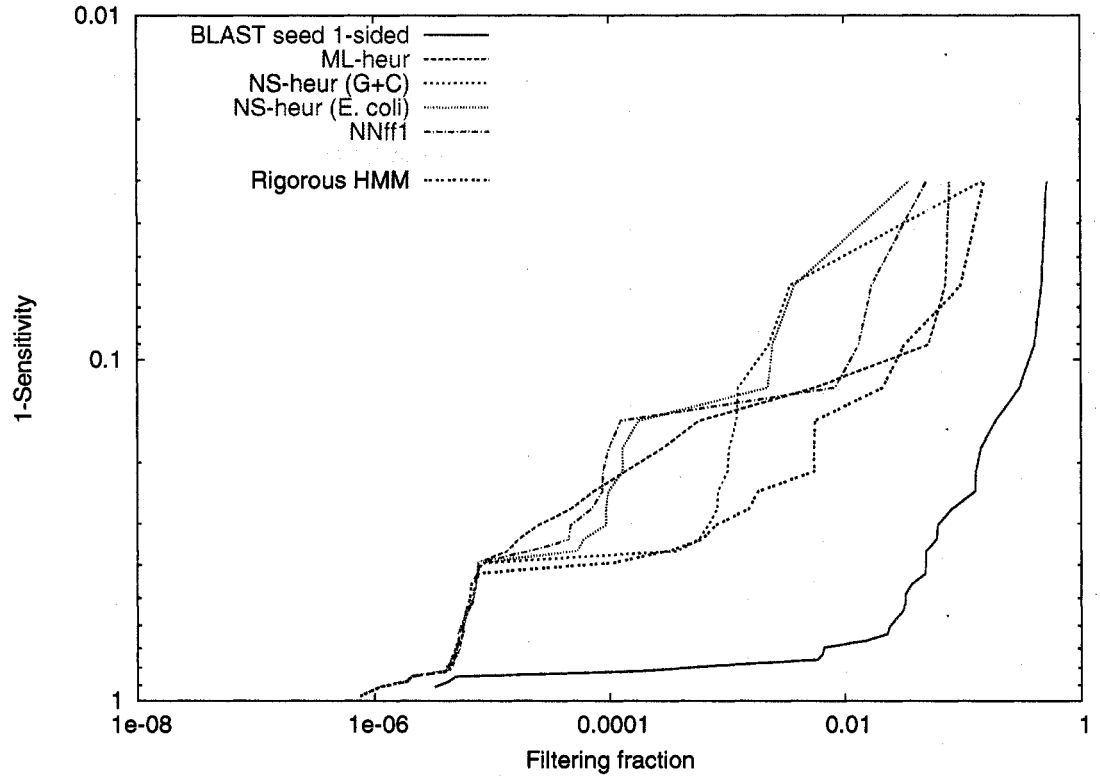


Figure 8.20: ROC-like curve: 6S-Ecoli (RSEARCH; 6S from Rfam family RF00013, on 1.2-gigabase microbe genome database) (log-scale sensitivity)

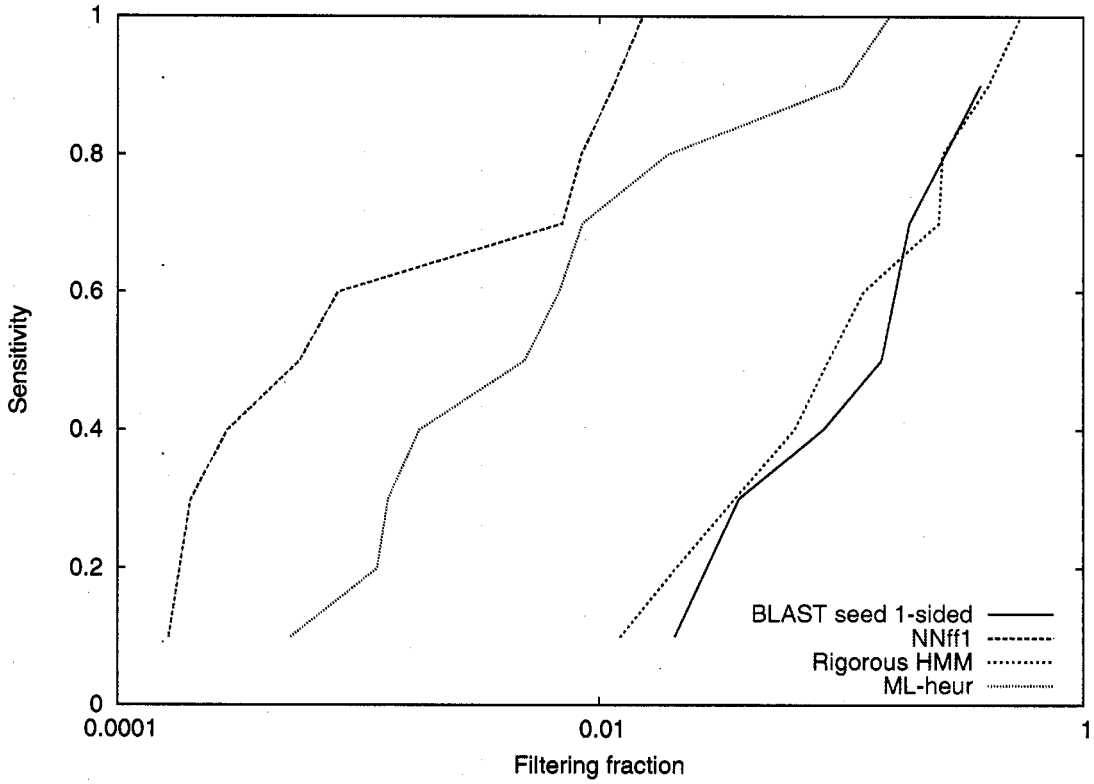


Figure 8.21: ROC-like curve: hs-srp-Arab (RSEARCH; human SRP on *Arabidopsis*)

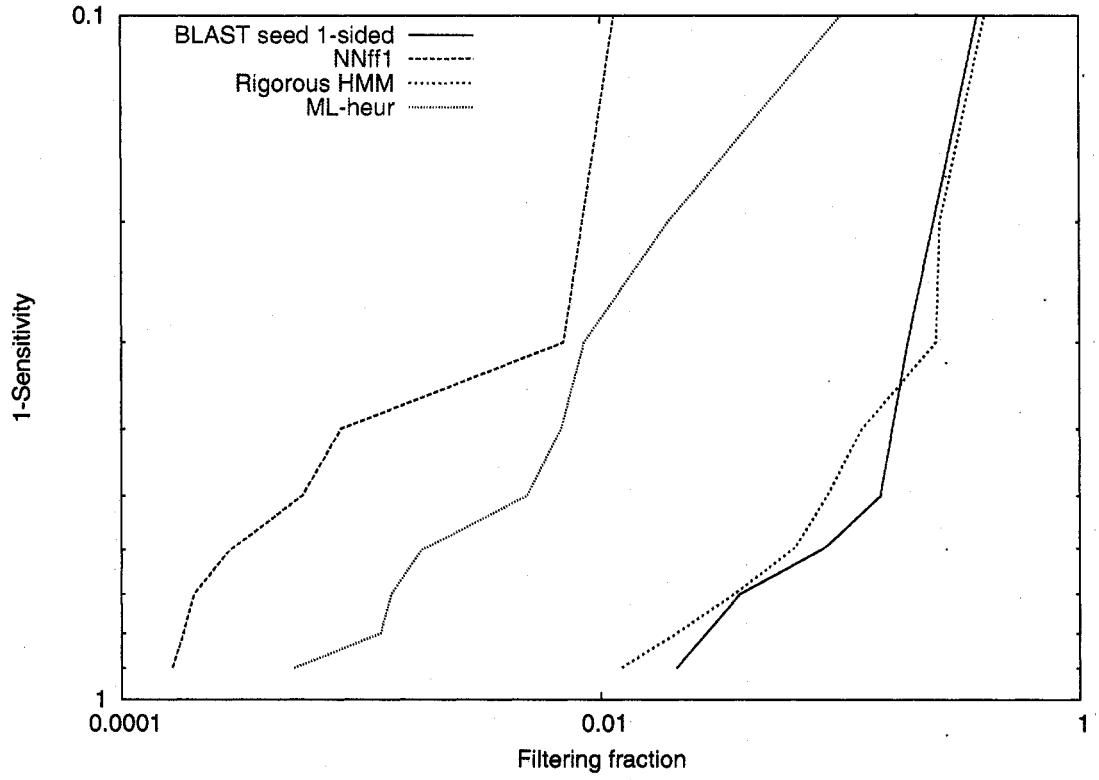


Figure 8.22: ROC-like curve: hs-srp-Arab (RSEARCH; human SRP on *Arabidopsis*) (log-scale sensitivity)

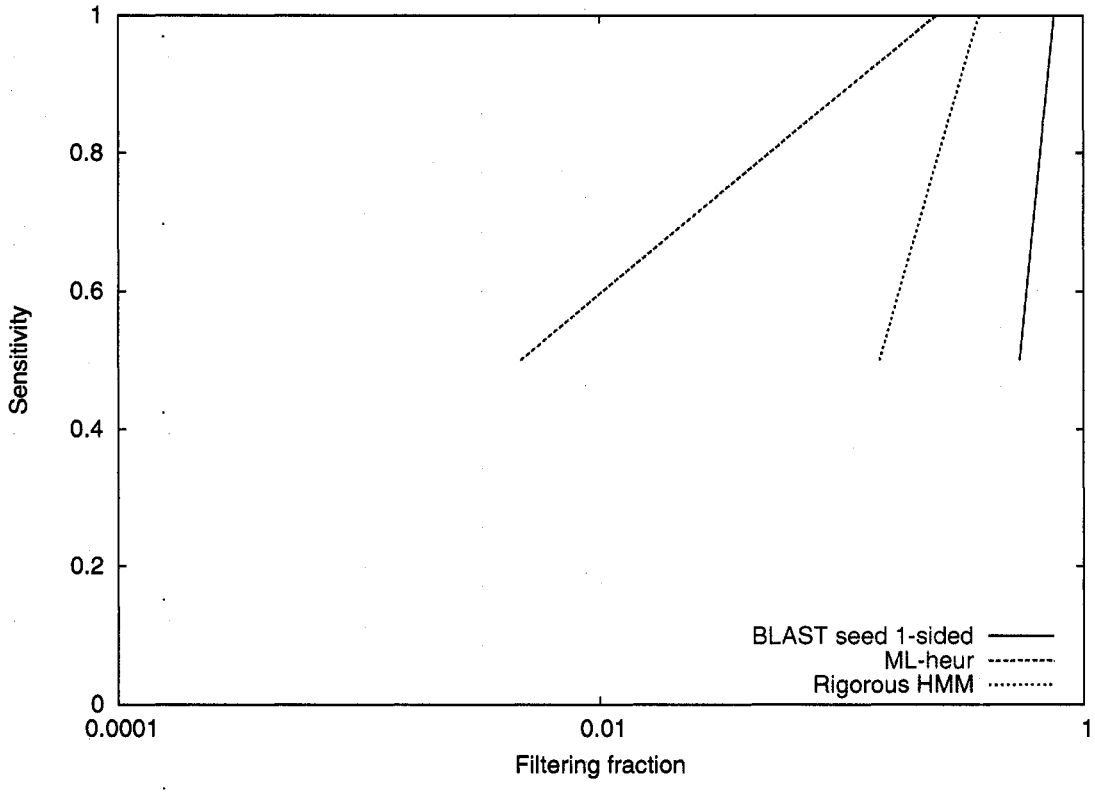


Figure 8.23: ROC-like curve: hs-srp-Arch (RSEARCH; human SRP on archaea)

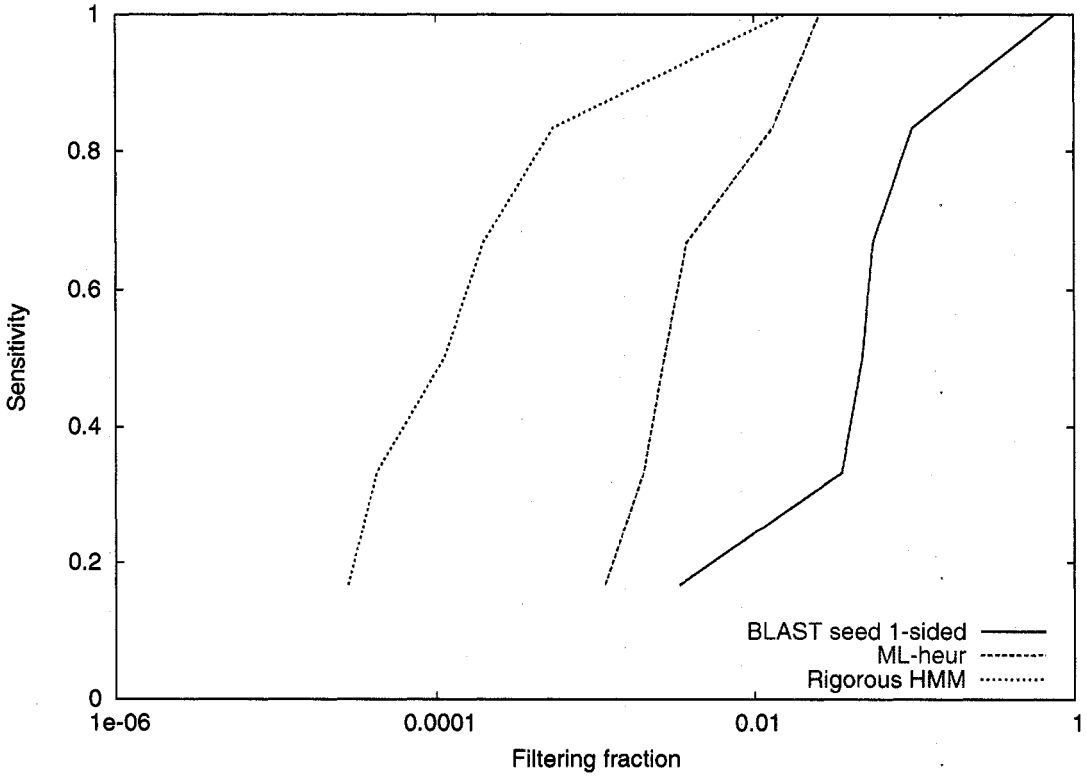


Figure 8.24: ROC-like curve: bs-srp-Arch (RSEARCH; bacterial SRP on archaea)

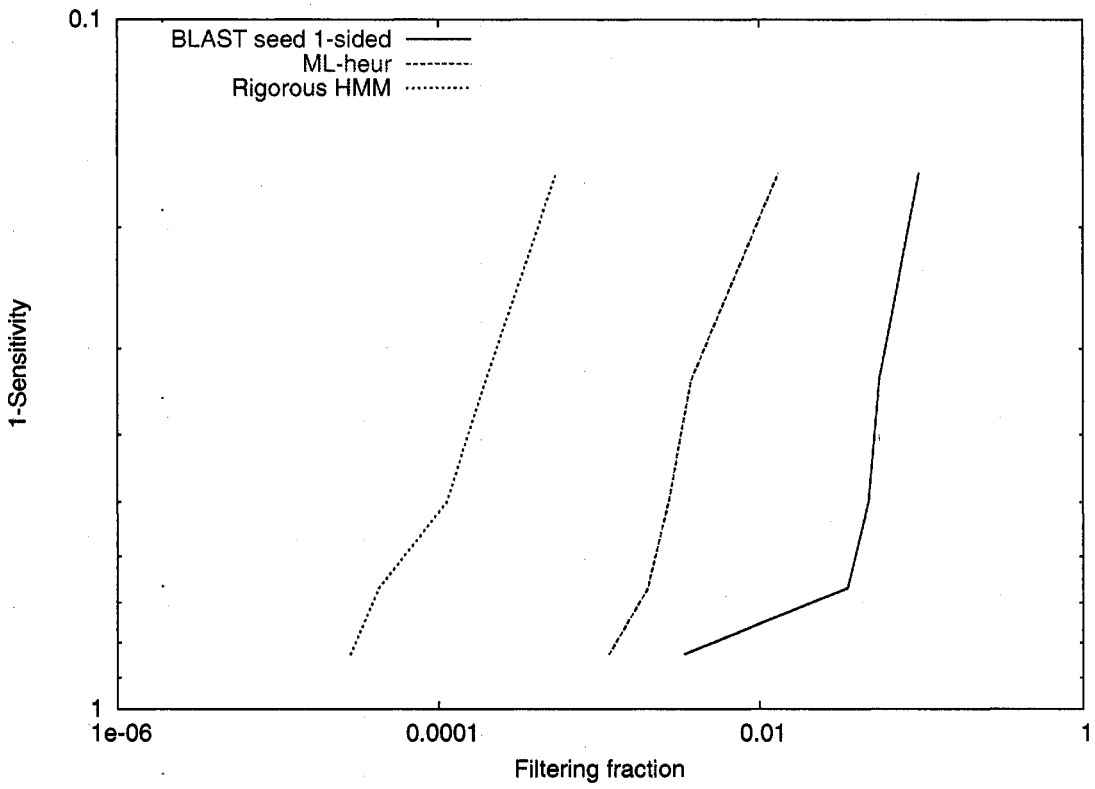


Figure 8.25: ROC-like curve: bs-srp-Arch (RSEARCH; bacterial SRP on archaea) (log-scale sensitivity)

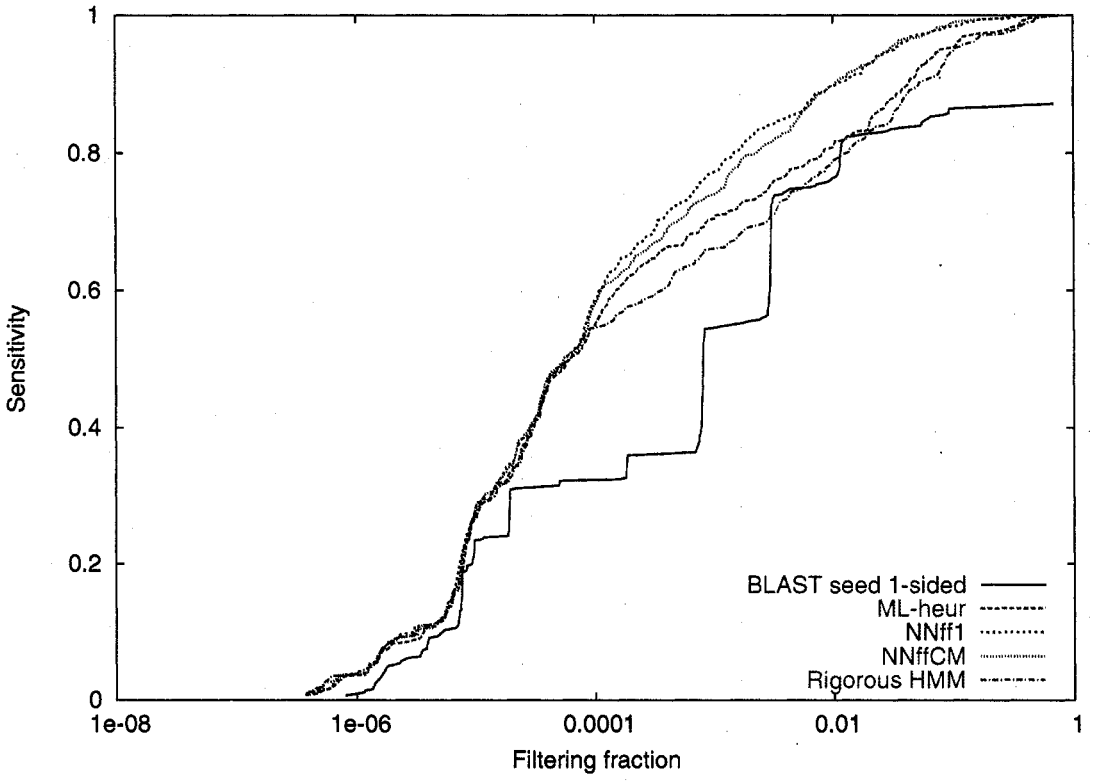


Figure 8.26: ROC-like curve: rf1 (RSEARCH; 5S rRNA from Rfam family RF00001, on RFAMSEQ)

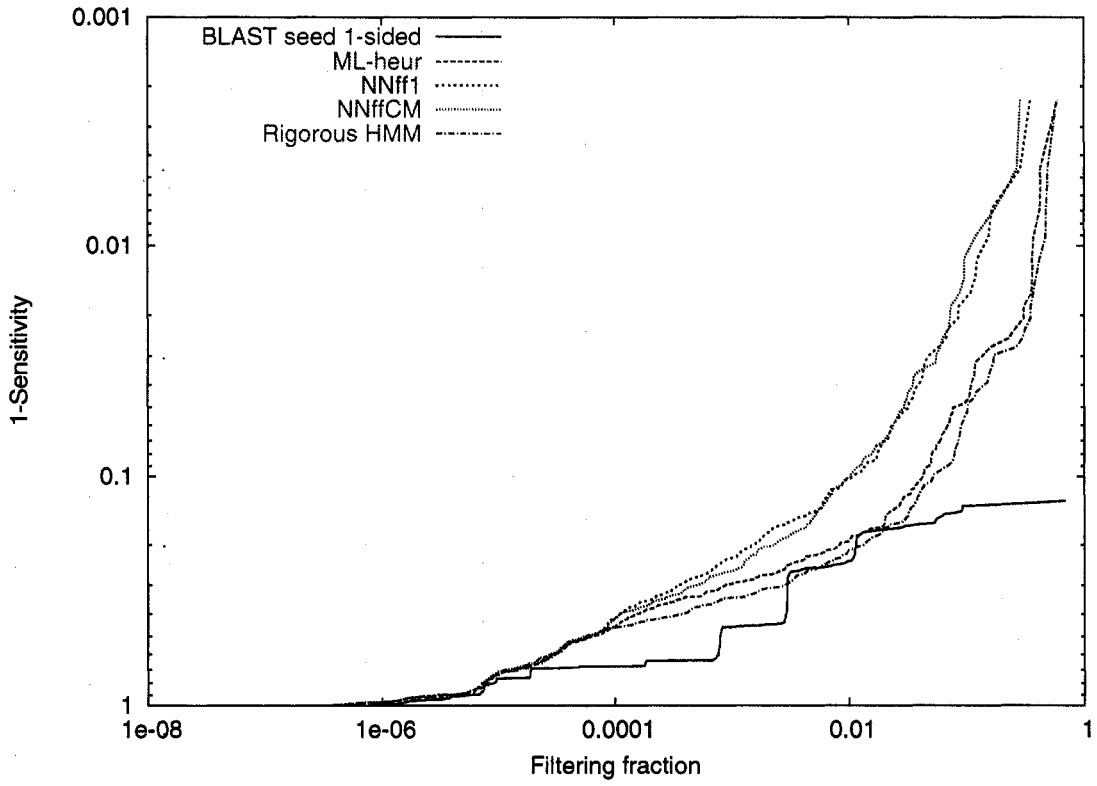


Figure 8.27: ROC-like curve: rf1 (RSEARCH; 5S rRNA from Rfam family RF00001, on RFAMSEQ) (log-scale sensitivity)

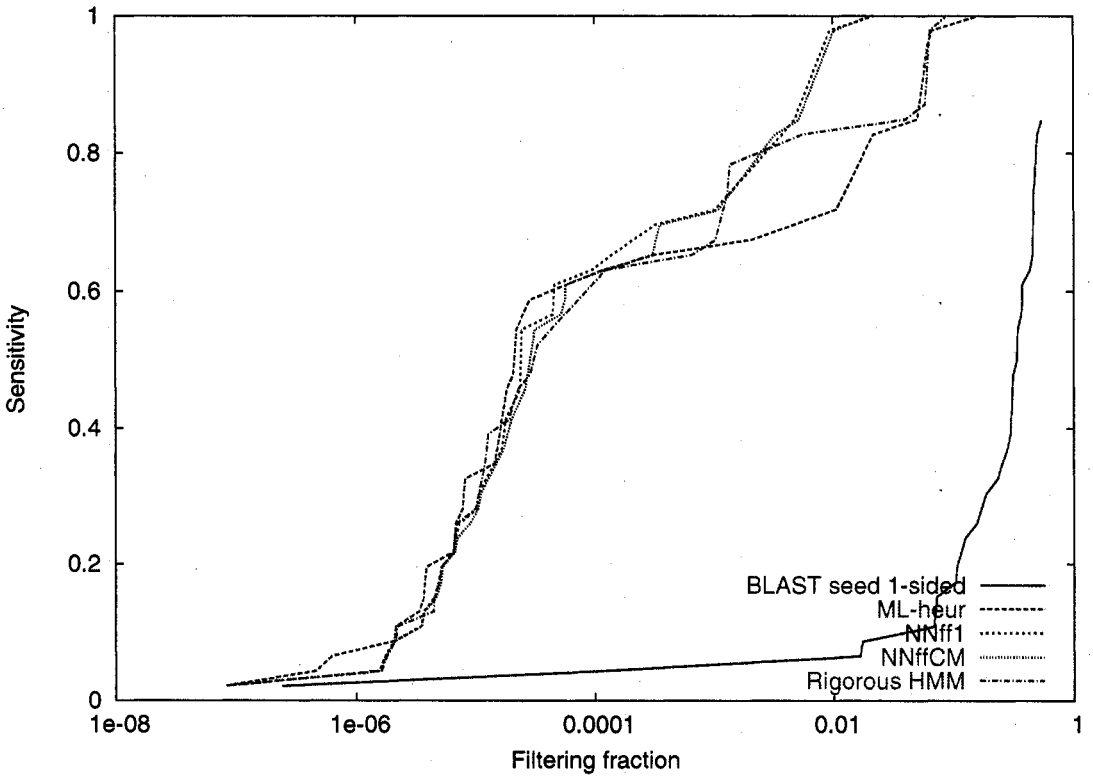


Figure 8.28: ROC-like curve: rf168 (RSEARCH; lysine riboswitch from Rfam family RF00168, on 1.2-gigabase microbe genome database)

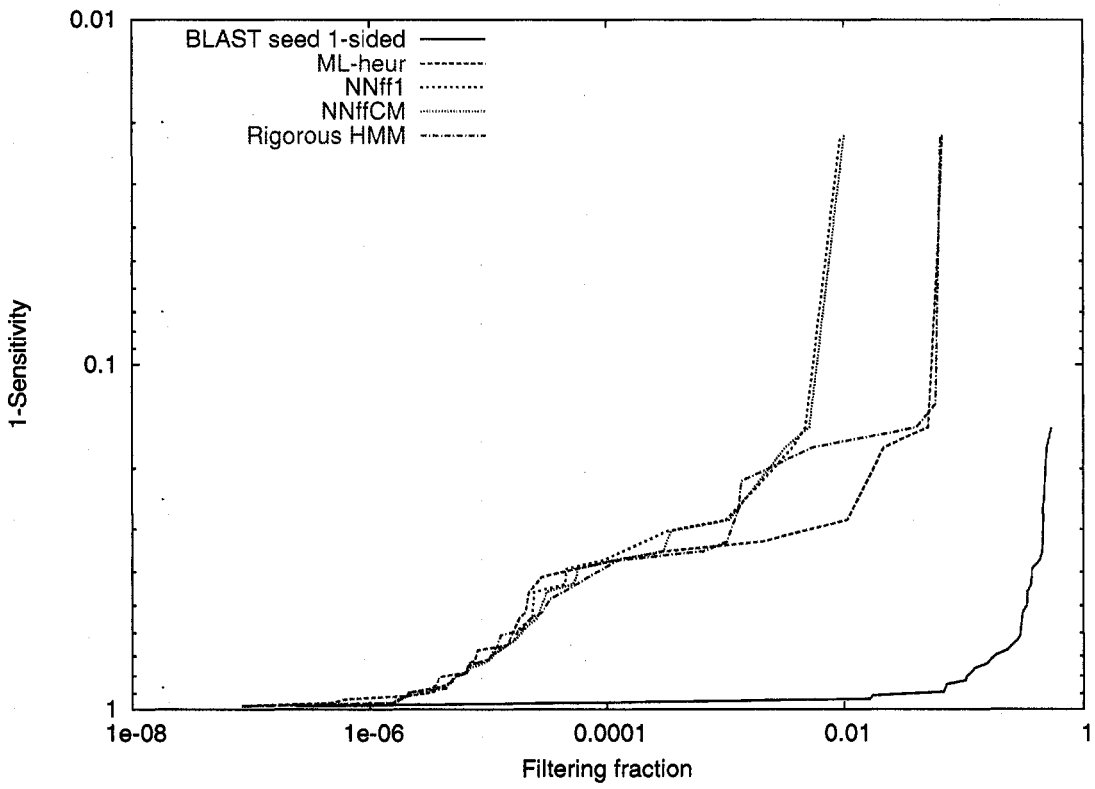


Figure 8.29: ROC-like curve: rf168 (RSEARCH; lysine riboswitch from Rfam family RF00168, on 1.2-gigabase microbe genome database) (log-scale sensitivity)

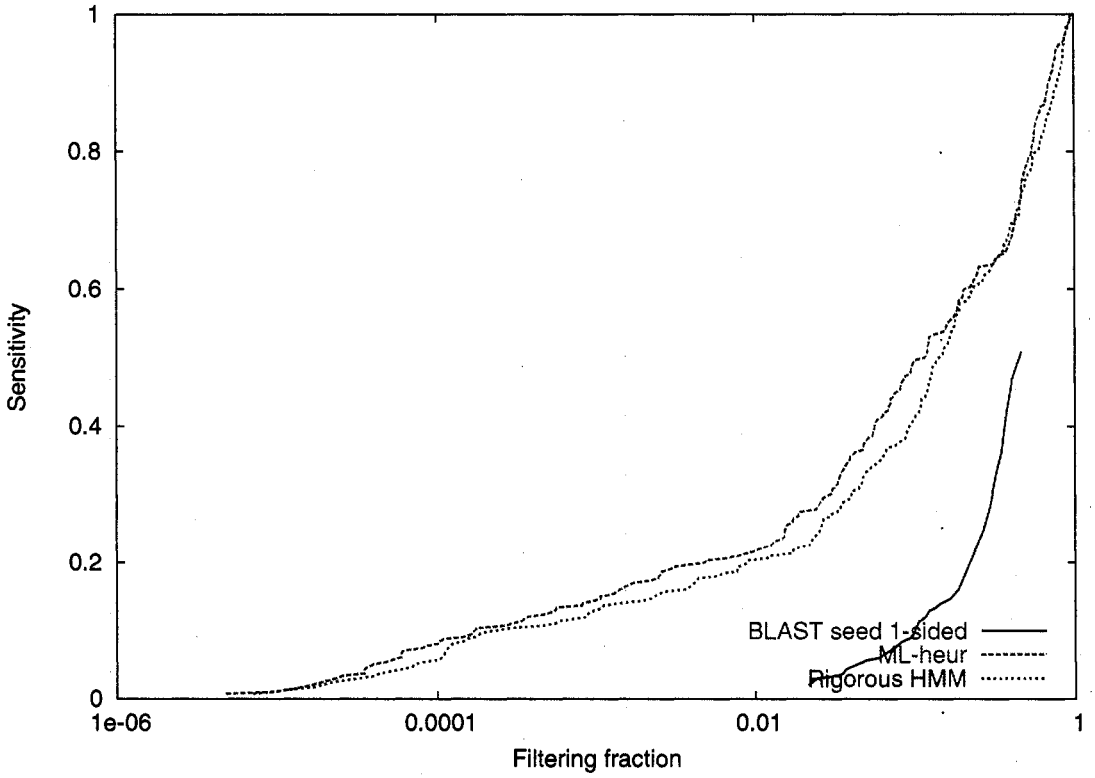


Figure 8.30: ROC-like curve: RF00005-Arch (RSEARCH; frog tRNA from Rfam family RF00005, on 1.2-gigabase microbe genome database)

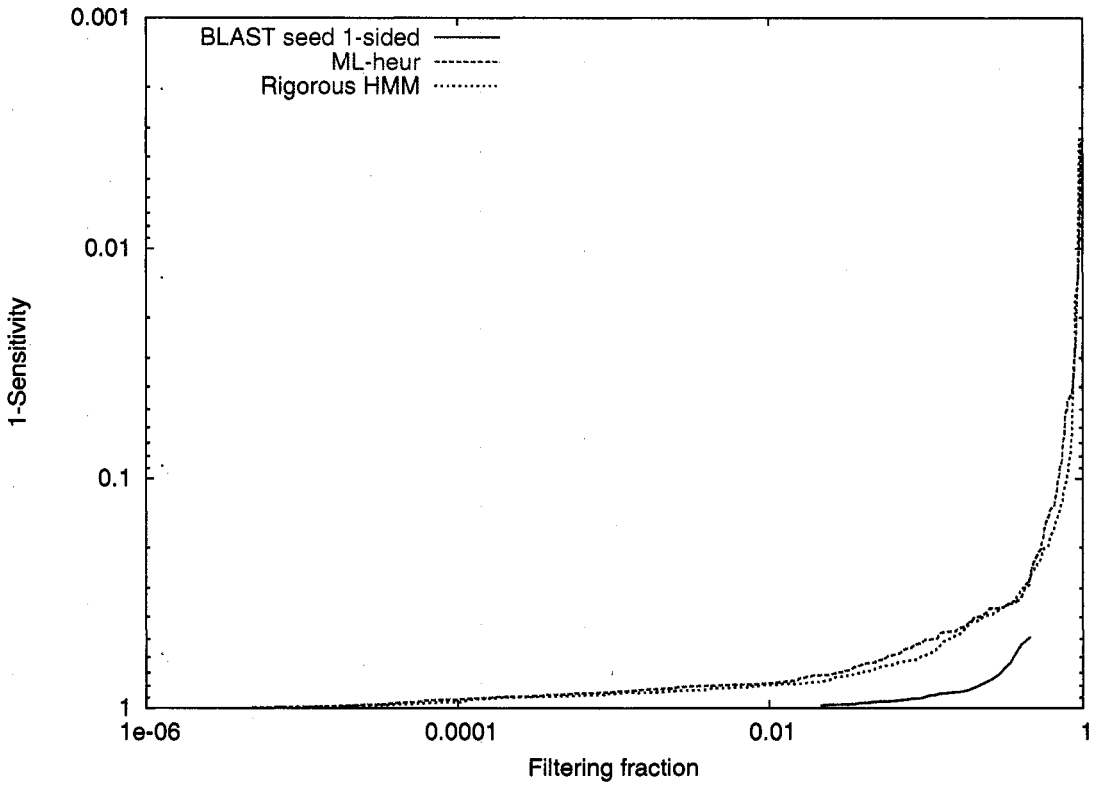


Figure 8.31: ROC-like curve: RF00005-Arch (RSEARCH; frog tRNA from Rfam family RF00005, on 1.2-gigabase microbe genome database) (log-scale sensitivity)

Chapter 9

DISCRIMINATIVE TRAINING OF HEURISTIC HMMS

This chapter describes a discriminative approach to learning parameters for a heuristic HMM. Discriminative learning uses positive and negative training data, and attempts to discriminate between them as well as possible. By contrast, probabilistic training algorithms such as the ML-heuristic and standard techniques like Baum-Welch [27] use only positive samples.

There are two main technical innovations in this chapter:

- A method to learn HMM scores discriminatively, based on positive and negative training samples. (The HMM grammar is identical to that of the ML-heuristic; only the scores are different.)

- A method to randomly sample hits with marginal E-values from a CM. (By "marginal E-values" , I mean that the hits are among the weakest we could detect with the given CM in a given database, e.g., E-values between 0.01 and 1.) The samples are used to generate positive samples for the discriminative training. Without random samples, many families have too few known members for discriminative training to be effective; indeed, for RSEARCH queries, only one family member is known. These random samples may be useful in many contexts.

Performance was generally superior to the ML-heuristic on a test dataset. However, it was not reliably better than the ML-heuristic when doing actual scans. I suspect that the technique works well for well-characterized families, but generalizes poorly when the RNA is not well characterized. Although performance is not reliable, the techniques in this chapter seem potentially useful in future work, so I present them here.

In chapter 7, I recommended running the ML-heuristic at a filtering fraction of 0.01. The discriminative heuristic in this chapter is designed to maximize the HMM's sensitivity

at a given filtering fraction (usually 0.01), which is exactly our goal. In order to efficiently solve the resulting optimization problem using techniques like gradient descent, I make the assumption that the positive examples' HMM Viterbi scores comes from a normal distribution. This assumption is roughly correct in practice, but is obviously an approximation. As I argue later, this normalcy assumption is not necessarily a debilitating one.

My technique most closely resembles Generalized Probabilistic Descent for Minimum Classification Error (GPD/MCE) [59]. GPD/MCE does not require normalcy assumptions, but rather minimizes an approximation to classification error. Unfortunately, this is a mismatch for our problem, since the filter will not perform a standard classification task; rather, at a filtering fraction of 0.01, the filter is forced to eliminate 99% of the sequence. Although I expect GPD/MCE would still be a reasonable heuristic, I did not investigate it for this reason.

Other techniques for HMM training include Maximum Discrimination (MD) [34] and the related Maximum Entropy (ME) [65]. Typically, these techniques are used to weight sequences, primarily to account for phylogenetic correlations. Since our training data is randomly—and independently—sampled from the CM, there are no phylogenetic correlations to worry about. Moreover, the techniques do not directly consider negative samples, possibly limiting discrimination.

I considered more general HMMs, possibly using state to model RNA base pairs better (e.g., chapter 5), or sequence analysis tools like conditional random fields [69]. In both schemes, modeling base pairs at variable distances (i.e., with indels) must reduce the filter's scanning speed; I wanted to see if it was possible to improve discrimination without reducing scanning speed. Also, like MCE/GPD, conditional random fields are not optimized for filtering.

There are numerous other discriminative machine learning techniques that could be applied. For example, support vector machines have already been shown to improve protein homology searches when trained on statistics extracted from a profile HMM [57]. I view these techniques as complementary to the techniques in this chapter. While techniques like SVMs should improve the ML-heuristic's performance, presumably they would do better given a better HMM.

All discriminative methods just described require positive samples. Unfortunately, of course, there may be few positive samples available for many families—especially in the context of RSEARCH, where only one positive sequence is available. So, all of these discriminative methods would presumably need some scheme to generate random positive samples.

9.1 Random sampling of positives

In this section I address the following problem. We are given (1) a probabilistic CM (in this chapter, I assume CMs are probabilistic, either by design, or by using the transformation scheme of section 8.2.1), (2) a database to be searched and (3) an E-value range defining marginal hits. Our goal is to generate independent samples from the CM that are within the E-value range on the given database. We are given a subroutine that outputs the E-value of any RNA sequence. (This subroutine is a CM scan of the sequence combined with E-value logic in RSEARCH [61].)

The rationale for this framework is that sequences whose E-values are lower than the E-value range are easy to find, so any filter will likely find them; sequences whose E-values exceed the range are not significant and will be obscured by false positives. The samples within the E-value range are therefore useful to train an HMM for filtering.

An unanswered question is: what should the distribution of the marginal hits be? I propose that the distribution should be equivalent to the following simple algorithm:

1: **repeat**
2: Sample a sequence x from the CM.
3: **until** the E-value of x is within the marginal range
4: Return x

To justify this, I assume that the probabilistic CM truly represents the probability of observing a given RNA sequence in a true homolog. Obviously the CM is an imperfect model, but it is the best model currently available, and has been successful in the context of finding RNAs.

If we accept this assumption, then sampling from the CM will generate a set of RNA sequences whose relative frequencies match our expectation of them being true homologs. The subset with marginal E-values are our best guess (given the probabilistic CM) of what true homologs with marginal E-values will look like.

However, the preceding algorithm would be very slow since it is very unlikely in general that sampling from the CM will generate sequences with marginal E-values.

9.1.1 A practical algorithm for marginal samples

Instead of sampling marginal E-values, I first consider the problem of sampling hits within a defined range of CM scores. Later, I show how to use this algorithm to sample hits with marginal E-values.

If we approximate CM scores to some accuracy a, a dynamic programming algorithm is enabled. This algorithm uses a table $T(\mathrm{S}_i, k)$, where S_i is a CM state, k is an integer and $T(\mathrm{S}_i, k)$ is the probability that a random parse starting at S_i would have a score in the range $[ka, (k+1)a)$. $T(\mathrm{S}_i, k)$ is sparse, since most values of k will not be needed; therefore, it requires a tractable amount of storage space. (Typically, $a = 0.01$, which yields a good tradeoff between accuracy and memory/speed requirements.)

Given such a table, sampling a random path is feasible. First, I show how to sample a random parse within the desired score range; from a parse, it is easy to generate the parse's sequence. For now, I ignore bifurcation states, but address them later.

Let $R(i)$ be the set of rules with state S_i as their left-hand side. For $r \in R(i)$, let $S(r)$ be the rule's score, $\Pr(r)$ be its probability and S_r be the state in the rule's right-hand side. (Since we are sampling paths from the CM, the nucleotides in the rule do not affect the probability and are irrelevant.) Finally, let $\pi^{\mathrm{S}_i \cdots}$ be the set of all CM parses beginning with state S_i.

Formally, then,

$$T(\mathrm{S}_i, k) = \sum_{\pi \in \pi^{\mathrm{S}_i \cdots}} \begin{cases} \Pr(\pi) & \text{if } ka \le S(\pi) < (k+1)a \\ 0 & \text{otherwise} \end{cases}$$

In words, $T(\mathrm{S}_i, k)$ is the sum of probabilities of all parses starting at S_i whose score is in

the interval $[ka, (k+1)a)$. This is equivalent to the earlier definition.

9.1.2 Sampling given the table

Suppose we wish to sample a parse starting at state S_i whose score is in the interval $s_l \ldots s_h$. (I assume $s_h - s_l \gg a$, so that the accuracy a is not too coarse.) Then the probability of choosing rule $r \in R(i)$ is proportional to

$$\Pr(r) \sum_{\pi \in \pi^{S_r \cdots}} \begin{cases} \Pr(\pi) & \text{if } s_l - S(r) \leq S(\pi) \leq s_h - S(r) \\ 0 & \text{otherwise} \end{cases}$$

(Later I normalize the proportional probabilities into actual probabilities.)

This formula sums over the probabilities of all parses that would remain in the desired score range. The formula bears a clear resemblance to the definition of $T(S_i, k)$. For example, suppose $s_l - S(r) = ka$ and $s_h - S(r) = (k+1)a$ for some k. Then the proportional probability of choosing rule r is $\Pr(r)T(S_r, k)$. If both $s_l - S(r)$ and $s_h - S(r)$ are integer multiples of a, then we can add up the adjacent cells in $T(S_i, k)$. If they are not integer multiples, we round them to the nearest integer multiple. This rounding reduces overall accuracy, but is a reasonable approximation. (Linear interpolation may do better, but I did not explore this.)

The necessity to add up multiple cells in $T(S_i, k)$ increases the run-time complexity of the algorithm. However, the overall run time is practical. For example, to find 10,000 random samples of a tRNA takes about one CPU hour.

At state S_i there is a set of rules $R(i)$ we can choose next in the parse. We know that one of these rules is chosen (since the parse must lead to the desired score range), so we normalize each probability, dividing by the sum of all the mutually exclusive probabilities.

This algorithm supports recursively sampling a parse, and therefore a sequence, using the following algorithm:

1: s_l and s_h are set by the user
2: $i \leftarrow 0$ {Starting state is S_0}
3: $\pi \leftarrow \emptyset$ {initially, the parse is empty}
4: **while** S_i is not an end state **do**

5: Randomly sample rule $r \in R(i)$ based on the current state S_i and score range $s_l \cdots s_h$

6: $s_l \leftarrow s_l - S(r)$ {adjust target score based on score of current rule}

7: $s_h \leftarrow s_h - S(r)$

8: $\pi \leftarrow \pi \circ r$ {append rule r to the parse}

9: $i \leftarrow$ the state in the right-hand side of rule r, i.e., the next state

10: **end while**

11: Output the sequence emitted by parse π

9.1.3 Choosing score ranges

I now relate score ranges to E-values. Since E-values and CM scores are closely correlated, it is reasonable to conclude that lower scores mean lower E-values and vice-versa. (In most applications, E-values are a monotonic function of score. However, RSEARCH-derived E-values vary based on G+C-content of the hit, although score and E-value are still closely correlated. My algorithm does not attempt to consider G+C content.)

Given the relationship between E-values and scores, I assume that there is some desired score range for which most hits will be in the desired E-value range. For simplicity, I fixed the difference $s_h - s_l$, by setting $s_h = s_l + 10$ (although other values can be used). The number 10 is chosen for two reasons: (1) if the number is too small, sampling a hit in the desired score range may be difficult—the scores obtainable by a given CM are sparse, and the approximations introduced by the accuracy parameter a make sampling a tight score range even more difficult, and (2) we do not want too large a range, otherwise the samples will have a vast range of E-values. In practice, the value 10 has not led to any problems, so there is no need to consider other values.

Given $s_h = s_l + 10$, it remains to determine s_l. First, we start s_l at some fixed value that can generate valid hits, e.g. $s_l = -10$. Then we generate a sample, find its score and then its E-value. If the E-value is too high, we increase s_l, and vice-versa. The amount by which to increase/decrease s_l is 1, which works well empirically.

(In practice, if the desired E-value range is 0.01 to 1, samples' E-values are usually between 10^{-4} and 10^{+3}.)

9.1.4 Building the table

I now show how to build the table $T(\mathrm{S}_i, k)$. (Again, bifurcations are dealt with later.)

$T(\mathrm{S}_i, k)$ is calculated in a bottom-up recursive fashion. In the base case, S_i is an end state, and $T(\mathrm{S}_i, k) = 1$ if $ka \leq 0 < (k+1)a$. (The score of an end state is always 0.) Otherwise, $T(\mathrm{S}_i, k) = 0$.

For the recursion,

$$T(\mathrm{S}_i, k) = \sum_{r \in R(i)} \Pr(r) T\left(\mathrm{S}_r, k - \left[\frac{S(r)}{a}\right]\right)$$

where S_r is the state in the right-hand side of rule r and $[x]$ rounds x to the nearest integer. The recursion does lose accuracy in the rounding.

9.1.5 Handling bifurcations

Bifurcations may seem to complicate the preceding approach because we must choose parses in each subtree independently; unfortunately, the scores of the independently selected sub-parses must sum within a desired range. So, I linearize the CM as follows.

Suppose we are faced with bifurcation rule $\mathrm{S}_i \rightarrow \mathrm{S}_j \mathrm{S}_k$, and we know that state S_j will eventually lead to end state S_{j^e}, i.e., $\mathrm{S}_{j^e} \rightarrow \epsilon$ and S_{j^e} is reachable from S_j. (If there are multiple end states, we create a new end state and call that S_{j^e}.) Ignoring local begins for now, there is no way in a CM to get into the states reachable by S_j without entering S_j.

To finesse the bifurcation, we (1) transform the bifurcation rule to rules $\mathrm{S}_i \rightarrow \mathrm{S}_j$ and $\mathrm{S}_{j^e} \rightarrow \mathrm{S}_k$, (2) modify grammar semantics so that the sequence emitted by S_k and its children will appear after whatever is emitted by states S_j through S_{j^e}. This change in semantics is inconsistent with context-free grammars, but while sampling parses, it is easy to keep track of these bifurcation rules, and move emitted subsequences appropriately. (By contrast, efficient parsing with the modified bifurcation semantics would be challenging.)

Local begin rules make it possible to enter a state reachable by S_j without ever entering S_j. Therefore, we make duplicates of S_j and all states reachable by it. The duplicate states are for the case where a local begin was used, and therefore S_{j^e} ends the parse and does not transition to S_k.

In theory, a maliciously designed CM could require $O(n^2)$ states (including duplicated ones) for an n-state CM. However, in practice few CM states are bifurcations and the two child states (S_j and S_k) generally have similar numbers of descendents. So, duplication increase the number of states by a factor of less than 3 in practice.

It is easy to accommodate bifurcations in calculating the table $T(S_i, k)$ given these grammar transformations. The transformed grammar rules can be used in the recursions already elucidated. It is necessary to disallow/require local begins appropriately under S_j, which requires some additional bookkeeping.

9.2 Discriminative training

A filter's goal is to eliminate sequences not containing homologs, while preserving the homologs for the CM to discover. Since filters are usually run at a constant filtering fraction that yields a desired trade-off between speed and sensitivity, it is logical to train the filter for this fraction.

In discriminative training, we are given positive training samples, i.e., homologous RNAs, and negative training samples, i.e., random sequences. Since we may have only a few actual positives, positives are randomly sampled from the CM.

The negative training sequence is randomly generated to match the G+C content of the target database; section 5.4.2 showed that G+C content is a significant factor in filter performance. This G+C-matched random sequence is generated using the method previously described for rigorous filters, in section 6.2.5.

The profile HMM's grammar rules are fixed, and identical to the ML-heuristic. However, its scores are determined by optimizing a discriminative training function.

The HMM assigns scores to sequences, both positives and negatives. Ideally, we want the positive scores to exceed all negative scores. For reasons described later, I assume that the HMM positive scores are roughly described by a normal distribution. For negative scores I use the empirical, maximum-likelihood distribution. (Later, I describe alternate assumptions.)

Given these assumptions, it is possible to calculate a single number representing the

predicted sensitivity (number of positives that would be discovered) at a filtering fraction of 0.01. The assumptions moreover permit calculation of partial derivatives to guide optimization. (Although the function is not strictly differentiable, these partial derivatives prove helpful in guiding optimization.)

9.2.1 Definition of scores

The definition of the positives' scores is straightforward: we run the HMM on each positive training sequence, and take the score of the last nucleotide, representing the best parse of the sequence.

For negatives, however, there is only one negative training sequence. Moreover, the HMM scores at each nucleotide position do not directly relate to filtering fraction. Instead of using HMM scores for negatives, I use inclusion points, as described in section 7.3. The inclusion points are the maximum score over all windows of size window length. The set of inclusion points can directly be used to find out what score yields a desired filtering fraction.

9.2.2 Calculation of mean and variance of scores

In the next section, I will define an objective function that uses normalcy assumptions, and therefore must estimate mean and variance of score samples. Estimated mean is equivalent to estimated expected Viterbi score.

Section 6.2.5 described a method to calculate the expected Viterbi score in the context of optimizing a rigorous filter. Briefly, a backtrace was done at each position, so that expected Viterbi score was $E[V] = \frac{1}{n}\sum S(\pi_i)$ where $S(\pi_i)$ is the score of the ith parse, π_i. Given rule score variables $l_1, l_2, \ldots$, this could be expressed as $\frac{1}{n}\sum k_j l_j$, where k_j is the number of times l_j's rule is used in the parses. Since the formula is symbolic, it can be differentiated. Even though it is not strictly differentiable (changes in l_j can change the Viterbi parses), it is effective at guiding an optimizer. For inclusion points, we use the parse backtrace corresponding to the highest-score in the window, i.e., the one that contributes to the inclusion point.

A similar scheme is used for estimated variance of scores. The standard estimator for

sample variance [116] is:

$$s^2 = \frac{\sum_{i=1}^{n}(S(\pi_i) - E[V])^2}{n-1}$$

This can be expressed using two terms. $\sum_{i=1}^{n} S(\pi_i)$ can be calculated based on $E[V]$.

The second term, $\sum_{i=1}^{n} S(\pi_i)^2$ is not calculated as a symbolic formula. All we require is the ability to evaluate this second term and to take its derivatives. Evaluating $\sum_{i=1}^{n} S(\pi_i)^2$ is straightforward. Its derivative is the sum of the derivatives of the $S(\pi_i)^2$. $S(\pi_i) = (\sum_j k_j l_j)^2$, where k_j is the number of times l_j's rule is used in π_i (usually k_j is 0 or 1, but $k_j > 1$ is possible if l_j is an insert state self loop rule.)

If l_x is some rule variable, then the partial derivative

$$\begin{aligned}
&\frac{\partial(\sum_j k_j l_j)^2}{\partial l_x} \\
=&2(\sum_j k_j l_j)\frac{\partial(\sum_j k_j l_j)}{\partial l_x} \\
=&2S(\pi_i)k_x
\end{aligned}$$

Thus, for each parse π_i, we need only accumulate $2S(\pi_i)k_x$ for each rule variable l_x.

The estimated standard deviation is simply the square root of the estimated variance.

9.2.3 A simple objective function

In this section, I develop a simple objective function that assumes that both positive scores and negative inclusion points are sampled from (separate) normal distributions. Let m_p, s_p be the estimated mean and standard deviation of the positives, and m_n, s_n for the negatives.

The filtering fraction 0.01 corresponds to the 99th percentile of inclusion points of negatives, which is $\approx$ 2.33 standard deviations. (Obviously, other target filtering fractions could be used.) Assuming inclusion points are normally distributed, the score threshold corresponding to filtering fraction 0.01 is $m_n + 2.33 s_n$.

We wish to maximize sensitivity at this score threshold. Sensitivity is the fraction of positive scores that are above the score threshold. Assuming positive scores are normally distributed, we can express sensitivity in terms of a Z-score. For example, for sensitivity of ~97.5%, the Z-score is -2, because a normal distribution has ~97.5% of its density greater

than 2 standard deviations below its mean. Because of the relationship between sensitivity and Z-scores, maximizing sensitivity at the threshold is equivalent to minimizing the Z-score at the threshold (but Z-scores are easier to work with). The Z-score corresponding to sensitivity is $Z = \frac{m_n + 2.33 s_n - m_p}{s_p}$.

Given the above normalcy assumptions, the discriminative training means minimizing the Z-score given the training data. Derivatives of the Z-score are calculated using the derivatives of means and variances; elementary calculates permits expressing the Z-score's derivatives in terms of m_p, s_p, m_n and s_n.

9.2.4 Removing normalcy assumption for negatives

I also investigated modeling the negatives using the empirical, i.e., maximum likelihood, distribution. It is easy to calculate the score threshold corresponding to a filtering fraction of 0.01 by sorting inclusion points, and taking the 99th percentile. However, it is less easy to take derivatives or guide optimization of this objective function.

To facilitate optimization, I make a weak assumption of normalcy, just for the purposes of obtaining derivatives. First, we empirically determine the score threshold at the desired filtering fraction, and continue to estimate the negatives' mean and variance. Using this mean and variance, we determine the Z-score of the given score threshold. If the negatives are normally distributed and the target filtering fraction is 0.01, this Z-score is ≈ 2.33.

If the score threshold's Z-score is z_n, then the overall objective function Z-score is $Z = \frac{m_n + z_n s_n - m_p}{s_p}$. Observe that the *value* of this objective function does not rely on a normal distribution for the negatives, but its derivatives do depend on normalcy. Since normalcy is likely to roughly resemble the actual distributions, these derivatives should guide optimization.

9.2.5 Alternate objective function assumptions

A variation on the above is to sample directly from the CM, instead of trying to sample marginal hits. For most CMs, these sampled scores will be very far from scores of negatives. Fortunately, with the normalcy assumptions, it is possible to work with these already

separate score ranges. Performance is similar, but not overall as good as when marginal samples are used.

The assumptions of normalcy in this chapter can reasonably be expected to select better HMMs over inferior ones, even though the true distributions are not normal—the selection of good HMMs is not as dependent on a perfect fit of distributions as, for example, estimation of E-values. However, the assumptions are theoretically unfortunate.

A better-founded distribution for HMM scores is the extreme value distribution (EVD) [27]. Unfortunately, estimation of an EVD from samples usually uses methods that cannot be written in closed form [28]. The EVD has simple formulas for its mean and standard deviation. Let the EVD be $\Pr(X) = e^{(a-X)/b - e^{(a-X)/b}}/b$ for parameters a, b. If the EVD's true mean and standard deviation are μ and σ, then $b = \sqrt{6}\sigma/\pi$ and $a = \mu - \gamma b$, where γ is the Euler-Mascheroni constant, $\gamma \approx 0.57721566$ [132], and $\pi \approx 3.14159265$. It is possible to obtain derivatives by first estimating normal distribution, and then using its mean and standard deviation to estimate a and b for an EVD. (The resulting estimates are not as reliable as for the standard methods.) This use of the EVD does not qualitatively change performance.

It may also be possible to use the minimum classification error (MCE) methods from the GPD/MCE technique earlier described [59]. In GPD/MCE, the max function of two variables x_1 and x_2 is approximated as $\max(x_1, x_2) \approx \sqrt[k]{x_1^k + x_2^k}$, for some constant k. A sufficiently large value of k will approximate the discontinuous max function, yet the approximation is differentiable. This type of technique may permit the use of empirical distributions for positive and negative scores, while still providing derivatives to guide optimization.

Another alternative is to restrict the freedom of variables. For example, instead of allowing all score variables to vary independently, I defined only two variables, b and e, where all local begins in the CM have score b and local ends have score e; other variables cannot change at all. This makes the discriminative algorithm less aggressive, so possibly safer. However, performance qualitatively resembled that when all variables can vary.

9.3 Results

The results I present here are preliminary and incomplete, since I became convinced that the techniques are not reliably better than the ML-heuristic in the context in which I wanted to use them. I discuss the following:

- Evaluation of the sampling algorithm.
- Analysis of ROC-like curves. (ROC-like curves were described in chapter 7.)
- Number of homologs found in actual scans.

9.3.1 On evaluating the sampling algorithm

I claim that samples generated as described are representative of actual homologs with marginal E-values. I compared CM-generated alignments of random samples versus actual homologs for some families, and found that they subjectively appeared similar. I evaluated similarity in terms of number of indels, use of local begins/ends and mismatched nucleotides.

Although this test is subjective, it is still useful. For example, a previous technique produced homologs whose CM-generated alignments were qualitatively different from those of true homologs. (In that technique, I raised CM probabilities to an exponent and re-normalized the rule probabilities for each state. When the exponent is 1, the CM is unchanged; exponents below 1 generate weaker hits since they tend towards a uniform model; exponents above 1 generate stronger hits; the exponent ∞ produces the consensus sequence, the strongest possible hit.)

Unfortunately, it is difficult to evaluate the samples quantitatively. For example, we might count the average usage of local ends within the sampled positives and separately within the true homologs. If these statistics are significantly different between the two sets, we conclude that the sampled positives are biased in some non-biological way.

However, the assessment of whether the statistics are significantly different is problematic because it is difficult to account for the fact that the true homologs will be highly correlated in a manner we cannot expect the CM to model. For example, the true homologs will

be correlated by phylogeny. It may be that there are a large set of biochemically feasible sequences for the desired activity of the ncRNA. However, due to phylogenetic correlation, only a narrow set of these possibilities may be explored. These will probably tend to use similar local ends (or other event) in the same places. Even a CM that correctly models the set of biochemically plausible sequences cannot hope to predict that this narrow set is the one selected. Therefore, even if the statistics are quantitatively different, we cannot conclude that the samples are poor representations.

In fact, it may be that sampled positives are too general to effectively train a filter. True homologs, being correlated by phylogeny and possibly other factors, will be concentrated in a narrower part of sequence space than sampled positives, and the CM may be too general to model the special case.

9.3.2 Analysis of ROC-like curves

I did ROC-like curves on full genomic data for some of the RSEARCH datasets in chapter 8; these curves are in section 8.4.2.

Positive training samples were generated by the sampling method described in this chapter, or by sampling directly from the CM. Sampling from the CM produces strong hits for the families tested here. (For some very small RNAs, sampling from the CM produces relatively weak hits.)

Negative training samples were usually generated using the G+C-matching random sequence algorithm of section 6.2.5. In some cases, a 0th-order model learned from the *E. coli* genome was used. (This is roughly a uniform model.)

The graphs use the following abbreviations:

- NNffCM = Positives sampled from CM directly, G+C-matching training sequence for negatives. Discriminative technique described here (positives assumed to be normal; negatives use empirical distribution, except normalcy assumed for derivatives).

- NNff1 = HMMs are first optimized as in NNffCM, then optimized again using marginal hit samples as the positive set. (The initial NNffCM phase is used to help convergence.)

- NSheur (G+C) = like NNffCM, but variant discriminative heuristic where both positives and negatives are assumed to be normal. G+C-matching negatives.

- NSheur (E. coli) = as previous, but *E. coli* genome used for negatives.

- NSheur (G+C, target) = like "NSheur (G+C)", but positive samples from marginal hits.

These experiments, being on a full genome database, are CPU intensive, so I have done relatively few tests. Tests that are not shown in the ROC-like graphs were not done.

I focus on NNff1, since that is the algorithm with the best theoretical justification. I tested NNff1 on four RSEARCH queries: *C. briggsae* 5S (Rfam ID RF00001), *E. coli* 6S (RF00013), a *B. subtilis* lysine riboswitch (RF00168) and human SRP (from RSEARCH paper). NNff1 and NNffCM were roughly tied for best on 5S (Figure 8.26) and the lysine riboswitch (Figure 8.28). NNff1 was best over most of the curve for 6S (Figure 8.19), although NSheur (E. coli) was very close. NNff1 was unequivocally best for human SRP (Figure 8.21).

9.3.3 Number of homologs found in actual scans

Based on the preceding results, I did RSEARCH scans to expand a batch of families, and tried both the ML-heuristic and the discriminative technique in this chapter ("NNff1" as per section 9.3.2). For each technique, I counted the number of hits with E-value less than 100, and with E-value less than 1. Hits with E-values above 1 are very marginal, but can be interesting. Hits with E-values less than 1 are most interesting, because it is fairly likely that they are true homologs.

These results, summarized in Table 9.1, were less favorable to the discriminative heuristic than the tests discussed in section 9.3.2. The ML-heuristic generally found more hits at E-value cutoff 100 than the discriminative technique. For an E-values cutoff of 1, the results were mixed: the ML-heuristic found more hits for two families, the discriminative found more for another two families.

When the discriminative heuristic found more homologs at cutoff 1, it found vastly more hits. However, this fact may simply stem from the greater overall number of homologs for these two families.

A concern with the cases where the discriminative heuristic is better is that two of the families, RF00168 and RF00230, are (1) relatively well characterized, and (2) two of the three families in the data set that are already known outside one bacterial group. (The other is RF00170, although this family is less well characterized.) This suggests that the discriminative heuristic may do better on well-characterized families, but my goal is to be successful for poorly studied, phylogenetically narrow families. In this context, the results suggest that the ML-heuristic is the safer choice.

9.4 Discussion

The discriminative technique works well for the test families, which were well-characterized, but its performance is not robust on poorly characterized families. There are four possible explanations for this phenomenon:

1. The results were unlucky; a larger test set may show the discriminative technique to be more generally superior than it appears.

2. The random positive samples may be biased in some way. Bias can arise either because of a fault in the sampling algorithm (e.g., the approximations turn out to be problematic), or may be a problem with the CM. If the CM describes the likely homologs poorly, samples generated from it will be biased. A CM may describe homologs poorly when it is wrong (e.g., the probabilities are not biochemically accurate) or too general (leaving the true homolog space as a tiny subset of what the CM proposes). Both problems are particularly troublesome in the context of discriminative training, because such training is more able to aggressively overfit the data than probabilistic methods such as the ML-heuristic.

   It may also be that the method to impose a probabilistic variant on RSEARCH CMs (see previous chapter) needs improvement. This algorithm worked well in the con-

Table 9.1: Discriminative vs. ML-heuristic on non-test RSEARCH queries. RSEARCH queries were run on the families in column 1 (out of the 147 queries in chapter 8) in order to discover novel homologs; i.e., these were not test families, but were families that seemed phylogenetically narrow. The discriminative method was the main one described in this chapter: "NNff1", as per section 9.3.2. The first column describes the family, with Rfam ID. As per chapter 8, an arbitrary family member was chosen to use as an RSEARCH query. The next two columns show how many hits were predicted by each method at E-value cutoff 100. "NNff1" is the discriminative, "ML" is the ML-heuristic. The winner (if any) is bolded. The next four columns show an E-value cutoff of 1. Whenever one of the techniques finds a hit that the other does not, the E-values of the hits are given. Exception: for RF00230 and RF00168, NNff1 finds many more hits, and these are not detailed. Note: for RF00168 and RF00170, I manually chose two family members, since both families include two distinct bacterial groups. Also, RF00168 (*B. subtilis*) is the test query described in chapter 8.

Family	**E<100**		**E<1**			
	ML	**NNff1**	**ML**	**ML: E-values**	**NNff1**	**NNff1: E-values**
RF00038 PrfA thermoregulator UTR	**15**	12	4		4	
RF00018 CsrB RNA	**78**	65	**22**	E=0.17, 0.79, 0.88	20	E=0.0060
RF00034 RprA RNA	23	**25**	15		15	
RF00040 RNase E 5' UTR	**34**	25	15		15	
RF00114 S15 leader	**40**	38	26		26	
RF00140 alpha operon RBS	**34**	32	28		28	
RF00168 (*E. coli*) lysine riboswitch	28	**42**	16		**24**	many
RF00168 (*B. subtilis*) lysine riboswitch	120	**123**	108		108	
RF00170 (*N. exedens*) retron MSR	1	1	0		0	
RF00170 (*S. enterica*) retron MSR	**85**	73	2		2	
RF00230 T-box	441	**612**	260		**351**	many
RF00236 plasmid copy control	31	**59**	13		13	
RF00238 plasmid copy control	**95**	67	25		25	
RF00242 plasmid copy control	27	**30**	**7**	E=0.015	6	
RF00378 Qrr RNA	**48**	44	20		20	

text of the ML-heuristic, but may not work well in the context of sampling. I have not significantly investigated discriminative performance for Rfam families (which use traditional CMs, which have a probabilistic interpretation).

I believe that the samples are likely a problem, since performance appears to relate to how well characterized a family is. Presumably, given a poor secondary structure, the RSEARCH CM has trouble generalizing to the correct distribution, and discriminative training is not always helpful. It is possible that better probabilistic RNA models would generate adequate samples.

Note that, even if the sampling algorithm is the problem, the samples may be useful in other contexts. Biases in samples can be magnified by powerful discriminative training algorithms, but these same biases may be less significant when only the broad properties of samples are important. For example, the samples could be used to predict a heuristic filter's rough sensitivity on marginal hits; biases here may not seriously undermine the estimates.

A potentially interesting test would be to use tRNAs, which have many known members, and compare (1) the ML-heuristic, (2) discriminative heuristic trained on random samples and (3) discriminative heuristic based on known tRNAs. If the last scheme does best, it supports the idea that the problem is the random marginal hits, either the sampling algorithm or that the CM is biased or too general.

3. The random negative samples may be biased. This seems unlikely to be a serious problem, but with powerful discriminative training, it may be that higher-order statistical effects are important; in this case, an HMM trained on 0th-order sequences, may generalize poorly to genomic sequences, which have complex correlations among nucleotides.

4. The discriminative algorithm's assumptions may be at fault. Although the assumptions of normalcy are clearly wrong, they are roughly correct. Moreover, the goal is merely to guide the optimization towards more discriminative HMMs, which is unlikely

to be highly sensitive to the exact distribution. (By contrast, slight discrepancies in the tails of a distribution can cause significant errors in E-value estimates.) Moreover, I experimented with a number of variations, all of which yielded qualitatively similar performance.

Given this, it seems unpromising to try GPD/MCE, which uses a similar framework, with different assumptions. Support Vector Machines (SVMs) may be more promising, since they are a radically different algorithm. However, if the random samples are a problem, the samples will also lead SVMs astray.

Chapter 10

EXAMPLE APPLICATIONS

I discuss three applications of my software:

- A collaboration with the Breaker Lab (Yale University), which led to the discovery of cooperative binding in a naturally occurring RNA [86], and the fact that 6S RNA is widespread among bacteria [6].
- Work expanding families RF00460 and RF00497, following on from chapter 8.
- A possibly improved SECIS element finder using techniques of chapter 7.

These experiments were all done using my software RAVENNA. RAVENNA is freely available under the GNU Public License [129].

10.1 Breaker Lab: cooperative binding RNA and widespread 6S

The Breaker Lab found, among bacteria, several conserved RNA-like patterns and associated MSAs [5]. I ran my software on alignments prepared by their group and analyzed results, although final evaluation of hits and incorporation into a new MSA was performed in the Breaker Lab.

Iteration of this process emerged as an extremely useful technique. After performing a search, likely positives were used to create a new MSA, and a search was done with the new MSA. Often the second search found more homologs.

This work led to two significant discoveries.

10.1.1 Glycine-binding riboswitch.

The Breaker Lab had identified 24 glycine-binding riboswitches in a variety of bacteria. They had identified two sub-types of the riboswitch, i.e., riboswitches within a sub-type are

less diverged than across sub-types.

Glycine riboswitches occur in tandem. After analyzing the results of a CM homology search using the ML-heuristic, I noticed additional homologs adjacent to most known homologs. In fact, roughly 80-90% of predicted glycine-binding riboswitches occur in tandem, typically separated by only 5-30 nucleotides. Moreover, the two sub-types corresponded to the first and second tandem riboswitch. Technically, what is in tandem is the *aptamer*, the actual RNA that binds glycine molecules. A separate *expression platform* is the part of RNA that turns on glycine-related genes when aptamers bind glycine. (The mechanisms of riboswitch expression platforms vary, but this one allows or disallows full transcription of the mRNA. In the absence of glycine, the mRNA is not fully transcribed, i.e., copied, from the DNA; in this case, protein cannot be made.)

Glycine riboswitches use cooperative binding. This raises the question: what is the glycine-binding riboswitch doing with two tandem aptamers? Experiments in the Breaker Lab demonstrated that the glycine riboswitch uses cooperative binding [86]. When one aptamer binds glycine, the other aptamer binds glycine much more readily. Results suggest that both aptamers must bind glycine in order to turn on gene expression.

Consider a curve plotting expression platform response as a function of glycine level. Suppose a more digital curve is desired, i.e., one that transitions more sharply from roughly "off" to "on". To achieve this, an aptamer can be tuned to respond more sharply. However, if the aptamer is at its limit, cooperative binding can help.

Cooperative binding is a well-known concept in biochemistry, and many proteins are known to use this strategy. To illustrate the basic idea concretely, I use a significant simplification. In my simplification, assume aptamer 1 is normal, while aptamer 2's affinity to glycine is dependent on aptamer 1: when aptamer 1 has not bound glycine, aptamer 2 cannot bind glycine at all, while aptamer 1's binding makes aptamer 2's affinity for glycine similar to that of aptamer 1. When glycine reaches levels that could turn aptamer 1 on, many aptamer 2 sites will become active, so there is suddenly more ability to bind glycine at the critical levels.

In reality, cooperative binding is not unidirectional; rather both aptamers 1 and 2 are less active at low levels, but when either binds glycine, the other becomes more active. Also, they are never totally inactive. However, in this more realistic scenario, the concept is similar.

Why use cooperative binding? The answer to why this riboswitch has two aptamers raises another question: why does it need cooperative binding? Many genes regulated by this riboswitch are involved in the glycine cleavage system. Although glycine is an amino acid used in proteins, it can also be used for energy. If glycine is plentiful, and particularly if other sources of energy are scarce, it is advantageous to use most available glycine for energy. However, if glycine is scarce, all of it must be used to make proteins, which are vital to the cell. These facts are a possible motivation for cooperative binding in this riboswitch: to carefully ration the use of glycine as circumstances change.

Cooperative binding is well-known among proteins. Some artificial RNAs have been designed to use this feature. However, the glycine-riboswitch is the first discovery of an RNA found in nature that uses this sophisticated mechanism.

10.1.2 6S is widespread in bacteria.

6S RNA was the first ncRNA to be sequenced—in 1971. Its function was only determined as recently as 2000. 6S is an RNA that binds σ^{70} RNA polymerase. RNA polymerases are protein complexes that copy genes' DNA into mRNA molecules. 6S can thus turn off genes with σ^{70} promoters, i.e., genes that use σ^{70} RNA polymerase for copying.

Even when 6S's function was known, it was only known to exist in a subset of γ-proteobacteria. One of the RNA motifs identified by the Breaker Lab was not a riboswitch, but was 6S RNA. The Breaker Lab's original RNA motif was found in Firmicutes—extremely diverged from γ-proteobacteria.

A variety of factors played a role in the realization that the motif was in fact 6S [6]. Although RNA homology searches did not find any homologs in γ-proteobacteria, searches did find homologs in other groups, which in turn led to other clues suggesting the RNA was 6S (though there were other lines of evidence unrelated to any computational work).

By iterating searches, most bacterial groups had predicted homologs, except the α-proteobacteria and *Aquifex aeolicus*. To find these, we created a database of the *Aquifex aeolicus* genome and two α-proteobacteria, and searched them with a local CM, using an MSA adjusted to accentuate an apparently unconserved area. The small database size provided better signal-to-noise, and permitted the CM to be run without filters, for additional sensitivity. Colleagues in the Breaker Lab evaluated hits, and the 5th-highest-scoring hit proved a likely homolog.

Two other groups independently discovered that 6S is greatly diverged outside of γ-proteobacteria [123, 133]. However, their detection of divergence was not comprehensive, and indeed BLAST was being used to detect homologs. One study did take advantage of the fact that 6S is typically upstream of *ygfA*, a protein-coding gene; homology searches for protein-coding genes are more reliable than those of ncRNAs. This improves the signal-to-noise ratio over BLAST, but risks a false negative if 6S fails to be upstream of *ygfA* in some cases. Indeed few β-proteobacterial and none of the Firmicute 6S homologs are upstream of *ygfA*. The CM-based homology search was accurate enough to discover the homologs without restrictively requiring *ygfA*. The other groups' approach was more experimental; 6S is extremely highly expressed, making it practical to identify in the wet lab. It was identified experimentally in β-proteobacteria, Firmicutes and *Aquifex*. All experimentally determined homologs matched our predictions.

10.2 Expanding families: RF00460 and RF00497

Based on significantly diverged homologs found by RSEARCH (see chapter 8), I chose two families and attempted to expand them fully. In both cases, I created an improved MSA that was used to update the Rfam Database. (This work was done while I was visiting Alex Bateman's group at the Sanger Institute in England.)

10.2.1 RF00460: polyadenylation inhibition element

U1A spliceosomal protein has a conserved RNA element in its 3' UTR called the polyadenylation inhibition element (PIE). The U1A protein binds PIE in its own 3' UTR, to regulate

Table 10.1: RF00460: original Rfam alignment. The "seed" alignment of RF00460, taken from Rfam 7.0. (The related Rfam CM finds one homolog in rabbit, but others are in mouse and human like the seed.) First column is the homolog's nucleotides in the form EMBL_accession_ID/start_nucleotide-end. Start is greater than end if the homolog is on the reverse strand. Next column is species. Third column has the actual sequences, and secondary structure represented using angle brackets.
Angle brackets (< and >) indicate base pairs, e.g., "<<.<..>>>" means that the first and ninth positions form base pairs, as do the second and eighth and the fourth and seventh. Columns are colored to (redundantly) indicate pairing; each half of a helix has the same color, so they can be matched up visually without studying the angle brackets.

	Species	**Sequence**
BC000405.2 /1511-1584	human	CCACAGAGGAUUGUACAAAAAGUGCUGUCCCCAAGAGAUUGCACAAAACGCUGUUUAGGCGGGAAUUAAAGUGGCUU
AC129561.13 /165007-165080	mouse	CCACAGAGGAUCGUACAAAAAGUGCUGUCCCCAAGAGAUUGCACAAAAUGCUGUUUAGGUUGUGAUUAAAGUGAGUU
L15447.1 /1070-1143	mouse	CCACAGAGGAUUGUACAAAAAGUGCUGUCCCCAAGAGAUUGCACAAAACGCUGUUUAGAUUGUGAUUAAAGUGAGUU
AC107835.7 /3781-3853	mouse	CUACAGAGAAUUCUAUAAAAAGUGCUGUCCAUAGGGUUUGCAUAAAACAUUGU.AGGGUGUAUUUAAAGUGGUUU
AC138288.13 /11165-11238	mouse	CCACAGAGGAUUGUACAAAAAGUGCUGUCCCCAAGAAAUUGCACAAAACGCUGUUUAGGUUGUGAUUAAAGUGAGUU
AC079644.9 /92689-92762	mouse	CCACAGAGGAUUUUACAAAAAGUGCUGUCCCCAAGAGAUUGCACAAAAGGCUGUUUAAAUUGUGAUUAAAGUGAGUU
		....<<<<<.......<<<<.<<<<<....>>>>>.......>>>>.>>>>>..<<<<<.........>>>.>>

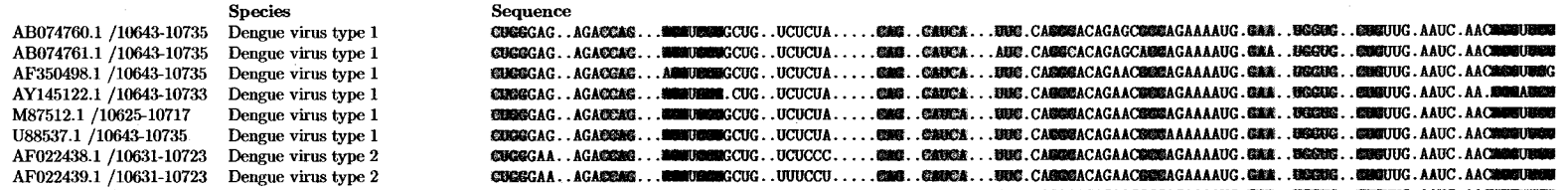
Table 10.2: RF00460: with new homologs. Filtered RSEARCH found the zebrafish homolog. Iterative scans leveraged this to find dog, frog and pufferfish. "Frog" refers to the African clawed frog, *Xenopus laevis*. "Pufferfish" is *Tetraodon nigroviridis*; its genome was downloaded for scanning from `www.genoscope.cns.fr/externe/tetranew/data/assembly`. The last helix in Table 10.1 did not appear conserved in the new sequences, so I removed it in this updated alignment.
(The column meanings, use of angle brackets and coloring is explained in Table 10.1.)

	Species	**Sequence**
BC000405.2 /1511-1584	human	CCAC.ACAGGAUUGUAC[redacted]A[redacted]..UCCC[redacted]AUUGCAC[redacted]C[redacted]UAGGCCGGAAUUAAAGUGGCUU
AC129561.13 /165007-165080	mouse	CCAC.ACAGGAUCGUAC[redacted]A[redacted]..UCCC[redacted]AUUGCAC[redacted]U[redacted]UAGGUUGUGAUUAAAGUGAGUU
L15447.1 /1070-1143	mouse	CCAC.ACAGGAUUGUAC[redacted]A[redacted]..UCCC[redacted]AUUGCAC[redacted]C[redacted]UAGAUUGUGAUUAAAGUGAGUU
AC107835.7 /3781-3853	mouse	CUAC.ACAGAAUUCUAU[redacted]A[redacted]..UCCA[redacted]UUUGCAU[redacted]CA[redacted].AGGGUGUAUUUAAAGUGGUUU
AC138288.13 /11165-11238	mouse	CCAC.ACAGGAUUGUAC[redacted]AG[redacted]..UCCC[redacted]AAUUGCAC[redacted]C[redacted]UAGGUUGUGAUUAAAGUGAGUU
AC079644.9 /92689-92762	mouse	CCAC.ACAGGAUUUUAC[redacted]A[redacted]..UCCC[redacted]AUUGCAC[redacted]G[redacted]UAAAUUGUGAUUAAAGUGAGUU
XM_533663.1 /1158-1231	dog	CCAC.ACAGGAUUGUAC[redacted]A[redacted]..UCCC[redacted]AUUGCAC[redacted]C[redacted]UAGGCCGGAAUUAAAGUGUUUU
BC064308.1 /1203-1278	zebrafish	CUUCA[redacted]AUUGCAC[redacted]A[redacted]G.UCUG[redacted]ACUGAAC[redacted]AGCUUUGUUGUUUUUAUUAAAGCAUGUU
Tetraodon chromosome 16	pufferfish	UCAC.[redacted]AUUGCAC[redacted]AA[redacted]GUUUAC[redacted]CAUUGUAC[redacted]AGCUUUUUCUUUUACAUUAAAACCUUAU
BC044979.1 /1100-1174	frog	UGUU.UCAGCAUUGCAC[redacted]A[redacted]C.AACU[redacted]AUUGUAC[redacted]AGCUUGUGUUGUUGUAAUAAACAUGAGA
		.....<<<<<.....<<<<.<<<<<.....>>>>>.....>>>>.>>>>>.....

Table 10.3: RF00185: original Rfam alignment. The "seed" alignment of RF00185, taken from Rfam 7.0. (The column meanings, use of angle brackets and coloring is explained in Table 10.1.)

Table 10.4: RF00497: original Rfam alignment. The "seed" alignment of RF00497, taken from Rfam 7.0.
(The column meanings, use of angle brackets and coloring is explained in Table 10.1. Colors do not always correspond to Table 10.3, since inferred structure is different.)

Table 10.5: RF00185/RF00497: with new homologs. Filtered RSEARCH and iterated searches beginning with RF00497 found several homologs, and showed that the family overlaps RF00185.
(The column meanings, use of angle brackets and coloring is explained in Table 10.1. Colors do not always correspond to Table 10.3 or Table 10.4, since inferred structure is different.)



its own polyadenylation; when the mRNA's normal polyadenylation is inhibited, it cannot be used to create a protein.

PIE homologs were known only in mammals (human, mouse and rabbit, in the Rfam entry). However, RSEARCH found a homolog in zebrafish, which suggested the family may be more diverged than suspected.

Since vertebrate genomes are large and the PIE element should be in the 3' UTR of U1A genes, I extracted sequences downstream of these genes to create a small database (using a program by Alex Bateman). Further searching revealed homologs in dog, as well as pufferfish (*Tetraodon nigroviridis*) and frog (*Xenopus laevis*). Using regular CMs with this restricted database proved more successful than my earlier strategy of using the zebrafish sequence as an RSEARCH query.

One helix in the annotated structure did not seem conserved, given the new families. I therefore removed it from the structure annotation.

The original alignment in Rfam 7.0 is reproduced in Table 10.1. My updated alignment is in Table 10.2.

10.2.2 RF00497: flavivirus element

RF00497 and filtered RSEARCHes on it were described in section 8.1.2 (page 150). Briefly, RF00497 is a conserved RNA element in the 3' UTR of the mRNA-like RNA viruses' genomes. I used an alignment of Dengue virus elements, and filtered RSEARCH searches found homologs in several other flaviviruses; iterative scans found the element in most sequenced flaviviruses, except those that do not infect mammals, or are not carried by mosquitoes or ticks. Scans also discovered that RF00185 was part of the same family.

These searches were done on the viral subset of RFAMSEQ, but did not require further restricting the database. Hits were easy to evaluate because true positives should be in a flavivirus (or at least a single-stranded, positive-strand RNA virus), and should be in the viral 3' UTR.

The alignment—which was originally based on Mfold predictions—had a number of isolated predicted base pairs, and was significantly changed given the new members. One

paper proposed a pseudoknot in the structure (based on indirect evidence) [113], but this pseudoknot appeared to not be conserved in the full set of RNA sequences. This paper, which identified the UTR element in the Japanese encephalitis virus group, suggested that a homolog exists in Dengue virus, based on similar melting curve features of the viral genomes. 3' UTR elements were known to exist in a variety of flaviviruses and evidence of structural conservation was known, e.g., [14]. However, no-one had found an alignment of the flavivirus 3' UTRs before the RSEARCH results, and I made the MSA without knowledge of results outside of Dengue virus.

Table 10.6 lists all fully sequenced flavivirus in the viral subset of RFAMSEQ I searched. ("Fully sequenced" means that the complete genome is available, to be sure that the 3' UTR is available. It is possible that some partial sequences include the 3' UTR, but that I did not notice them. It is also possible that an apparently complete genome missed the 3'-most nucleotides; the RF00497 element is very close to the 3' end.)

RF00497 was found in all tick- and mosquito-borne flaviviruses that infect mammals, except for Alkhurma virus. I do not know why a homolog was not found in Alkhurma.

Other flaviviruses with no predicted homologs fell into two classes: (1) viruses that infect mammals, but have no known vector (i.e., no organism like tick or mosquito is known to carry it), or (2) viruses that infect mosquitoes only, but are not known to infect mammals. Previous work discovered elements in some flaviviruses without known vectors: Modoc virus, Rio Bravo virus and Montana myotis leukoencephalitis virus [19]. I integrated these predictions into my MSA, and they appear homologous, based on conserved structure and sequence elements and the fact that they are in the appropriate region of the viral genomes. However, the homologs were significantly diverged, and many short motifs that appeared conserved in the other flaviviruses are modified in the no-known-vector case. No Yokose virus homolog was found, either in previous work or by additional CM scans.

Although CM scans missed the no-known-vector flavivirus homologs, these homologs were found using a labor-intensive process, in which specific regions of the viral genomes were carefully analyzed. By contrast, the CMs scanned a much larger viral genome database, and used no additional analysis. For the flaviviruses lacking a predicted homolog, either these flaviviruses truly have no RF00497 homolog, or it is too diverged for the CM (or filter)

Table 10.6: Flaviviruses and RF00497. This table lists all flaviviruses whose complete genome was available in the viral subset of RFAMSEQ that I searched. The first two columns give the viral group and species name. The third column classifies the vector (what organism carries the virus), by [19, 23]. Most viruses infect mammals or birds, and are carried in mosquitoes or ticks. "NKV"=no known vector. "Mosquito only" means the virus is not known to infect vertebrates. In the last column, "Found" means a convincing homolog was found, "Found (*)" is the same, but the homolog was not put into the MSA since it was similar to another sequence in the MSA, "None" means no homolog was predicted. Viruses that do not appear to have complete 3' UTR sequences available are not listed.

Viral group	**Virus**	**Vector**	**Found?**
Dengue	Dengue	Mosquito	Found
Japanese encephalitis	Japanese encephalitis	Mosquito	Found
	Murray Valley encephalitis	Mosquito	Found
	Usutu	Mosquito	Found (*)
	West Nile	Mosquito	Found
	Kunjin	Mosquito	Found
Modoc	Modoc	NKV	None
Ntaya	Yokose	NKV	None
Tick-borne encephalitis	Alkhurma	Tick	None
	Deer tick	Tick	Found (*)
	Langat	Tick	Found
	Louping Ill	Tick	Found
	Omsk hemorrhagic fever	Tick	Found
	Powassan	Tick	Found
	Tick-borne encephalitis	Tick	Found
Yellow fever	Yellow fever	Mosquito	Found
Unclassified	Cell fusing agent	Mosquito only	None
	Kamiti River	Mosquito only	None
	Montana myotis leukoencephalitis	NKV	None

or manual effort to find.

Original Rfam alignments for RF00185 and RF00497 are in Table 10.3 and Table 10.4, respectively. Table 10.5 is my updated alignment.

10.3 SECIS element finder

The results of chapter 7 showed that the ML-heuristic is 9 times more sensitive than BLAST in finding SECIS elements (Rfam ID RF00031), which suggested that filtered CM solutions may help to build a better SECIS element finder.

The SElenoCysteine Insertion Sequence (SECIS) element directs the incorporation of the "21st" amino acid selenocysteine. In eukaryotes, when an mRNA includes a SECIS element in its 3' UTR, UGA codons code for selenocysteine residues, instead of their usual STOP function. Proteins with selenocysteine residues are called *selenoproteins.*

Selenoprotein annotation is important for two reasons. First, due to the unusual use of the UGA codon, current protein-coding gene prediction programs cannot hope to correctly predict selenoproteins' exons.

Second, selenoproteins have medical significance in that they are the major context in which cells use selenium [122, 66]. There is strong evidence that selenium deficiency causes male infertility specifically due to compromised function of a selenoprotein, PHGPx, that functions in sperm development [85]. Less direct evidence suggests possible roles for selenoproteins in, for example, cancer [40, 121] and brain function [112].

Recent work screened various organisms for selenoproteins, seeding searches based either on a predicted SECIS or on conserved ORFs with internal UGA codons, and conservation continuing after the UGA [67, 18, 68]. Improved SECIS element prediction may help to discover new selenoproteins, and to relax constraints elsewhere in the pipeline.

Previous work on detecting selenoproteins has used the program SECISearch [67]. Basically, this program uses the PatScan program [26] to detect a SECIS-like pattern, then folds candidates using Mfold [140] using an energy threshold and imposing other constraints on the predicted structure to reduce the false positive rate.

I compared CMs with ML-heuristic filters on Rfam family RF00031 to SECISearch 2.1.

To assess sensitivity, I used 61 GenBank entries containing known selenoproteins in the SelWorld Database (`http://www25.brinkster.com/selworld/index.asp`). Many entries in this database were undoubtedly used directly or indirectly in both SECISearch and in the Rfam SECIS family, but some sequences are new. To assess selectivity, I ran both programs on random sequences generated with uniform, independent probabilities of A, C, G and T. For robustness, I estimated false positive rates based on at least 10 predictions in random data. (The sequences tested were 5 Kbases. I tested sequences until SECISearch had predicted SECIS elements in 10 sequences. I ignored multiple predictions in the same sequences, since typically these are redundant hits with slightly different nucleotide positions.)

Under default settings, SECISearch found SECIS elements in 30/61 SelWorld sequences. It predicted 1 false positive per 0.6 Mbases. Filtered CMs found SECIS elements in 34/61 SelWorld entries, and predicted 1 false positive per 16.7 Mbases. I also tested a "looser" SECIS pattern supplied with SECISearch 2.1. With this setting, SECISearch found 35/61 SECIS elements in SelWorld, but its false positive rate was much worse—1 per 0.036 Mbases—than with its default pattern.

I conclude that at comparable levels of sensitivity, the CM solution is over 20 times more selective than SECISearch 2.1 in its default settings. This preliminary experiment is somewhat crude, e.g., variation in C+G content is not considered, nor is a more realistic random model of genome sequences. However, it is difficult to explain a factor of 20 difference by such issues.

It would be promising to integrate a CM-based SECIS element finder into a selenoprotein detection pipeline.

Chapter 11

FUTURE WORK

There are three types of future work I discuss here:

- **Improvements to filters for ncRNA searches.** This is the most direct extension to my dissertation work.
- **Applications of my algorithms.**
- **Improvements to ncRNA search algorithms.** This dissertation improved CMs by making them faster. However, there are other ways to improve them.

11.1 Improvements to filters

11.1.1 Rigorous filters

Although rigorous filters work on the vast majority of Rfam families (chapters 4, 5 and 6), they do not work on all. In the non-local case, the SECIS element is biologically important, and it would be nice to have a technique that supports it. For the local CMs, several families cannot be rigorously filtered in practical time. The sub-CM technique may not work well for local CMs due to its reliance on short hairpin-specific window lengths. The store pair may need to be extended to allow memory of local begins. Or other techniques may be required.

A* search. One alternate scheme for designing a rigorous filter is to adjust the dynamic programming algorithm to prune out partial parses that cannot score above threshold [31]. Such a strategy is related to A* search [109], although usually such techniques are applied to problems believed to be NP-complete or harder, where SCFG parsing has a polynomial-time solution. Search strategies require a function that takes a partial parse and returns an upper

bound on the score of the full parse, as well as a heuristic to decide which step to explore next in the parse. A theoretical advantage of this strategy is that it may benefit significantly from even a loose upper bound function, whereas rigorous filters that can "almost" eliminate sequences are no different from those that fail completely.

I implemented a scheme based on this, but was only able to eliminate about 50% of the dynamic-programming table. (Moreover, I was testing on a family, MicF in Rfam 4.1, whose RNAs are highly similar in sequence; thus, this family is very easy to filter.) However, my upper bound function was the highest possible score that a subsequence of the given length could have; a better function may improve results. Using the rigorous HMM is a tempting prospect, although the most obvious implementation would require re-starting the HMM Viterbi algorithm at every nucleotide (since the HMM Viterbi algorithm does not directly deal with substrings like the SCFG algorithm does), which would be slow. However, more clever schemes may be possible.

E-values, G+C content, ... As CM techniques are improved, designing rigorous filters may become increasingly challenging. The RSEARCH program has introduced E-values for CMs. It turns out to be necessary to consider the G+C content of hits in order to calculate accurate E-values; CMs can be heavily biased towards low or high G+C (e.g., section 5.4.2), and it may therefore be easy to obtain a high score with say a 5%-G+C sequence. RSEARCH calculates the G+C of a hit and looks up the appropriate E-value statistics for the given score and G+C content; this is already non-trivial for the rigorous filters I presented. Obviously, other improvements to CMs are possible, and these could also make rigorous guarantees challenging.

Rigorous filters on other problems. It is common practice in computational biology to design heuristics to speed algorithms that would otherwise be impractical. A key advantage of rigorous speed-ups is that they eliminate the question of whether a better algorithm would have found more results. It is possible that techniques like those in this dissertation will suggest applications to other domains.

11.1.2 Heuristic filters

Add structure. The heuristic filters I have developed use no structural information; clearly the addition of structural information would likely improve their accuracy. One approach would be to apply the store-pair and sub-CM techniques of chapter 5 as heuristics. A problem is that powerful store-pair or sub-CM filters are very slow, so it is crucial to apply the filters in series.

Unfortunately, in the heuristic case, it is necessary to consider the filters' sensitivity in designing series of filters. (This problem does not arise for rigorous filters, since they are guaranteed to have 100% sensitivity, so their composition must also have 100% sensitivity.) Predicting the sensitivity of heuristic filters and of compositions of filters is an open problem that would have to be solved to apply this strategy.

A different approach is to consider pattern-matching ncRNA predictors as filters. Although these approaches have limitations, they may be adequate as filters. The pattern-matching algorithms often have high theoretical complexity, but run quickly in practice. One of tRNAscan-SE's heuristic filters uses a tRNA pattern [38]. Pattern-matching filters may work well if patterns can be automatically determined from the CM. This will require computationally determining what are the most significant features in the CM, since a pattern based on the whole CM is likely to be prohibitively slow.

Support vector machines. Previous work has improved profile HMM-based homology search for proteins using Support Vector Machines (SVMs) [57]. The SVM was trained to discriminate based on the HMM's *sufficient statistics*, roughly the posterior probabilities that each rule is used in parsing a given protein sequence. This use of SVMs improved homology search accuracy over profile HMMs.

It may be non-trivial to extend this method to the RNA homology search. The method requires probabilistic models, but local ends are not well defined probabilistically since they effectively have probability 1 of matching any length of subsequence. However, a probabilistic equivalent could be established.

Another issue is that in the protein problem, database sequences are full-length protein

sequences, and we match globally to the sequence. For RNA, database sequences are typically chromosomes or genomes, and we wish to find a small subsequence that is an RNA. This presents two challenges. First, the use of LOD scores in the profile HMM is necessary since otherwise any insertion is penalized [27, p. 51]. When using a probabilistic model, we must somehow make sure it is not overly penalizing insertions.

Second, and more seriously, calculating the sufficient statistics requires the Forward and Backward algorithms [57, 27]. The most natural implementation is to do this for the full chromosome or genome sequence, but this is space inefficient and moreover is difficult to interpret in terms of a specific predicted RNA. Basically, we do not ask if the chromosome contains an RNA (a bad question, since the signal is likely diluted), but whether a small subsequence is an RNA. Global sufficient statistics would need to be adapted to this local case.

A different problem for SVMs is that we may not have many positive training sequences available; in the RSEARCH case, we may have only one. To generate positive samples, it may be reasonable to use the method in chapter 9 to sample marginal hits directly from the CM. Although this sampling method was not effective in the context of the discriminative filters I considered, it may be useful in this context.

This use of SVMs is related to postprocessing the HMM's Viterbi parse. Inspection of the Viterbi parse may reveal that the profile HMM aligned matching base pairs, even though there is no guarantee that the sequence-only profile HMM will do this. If profile HMM Viterbi alignments are approximately correct, we may derive extra information by checking for matching base pairs. In fact, the SVM may be able to implicitly check this, since the sufficient statistics will indicate probabilities of matching base pairs.

Filters that scan faster. When the ML-heuristic is used with a filtering fraction of 0.01, as I recommend, the CM dominates overall scan time. However, if a more accurate heuristic is designed, a lower filtering fraction becomes preferable, and then the filter will be responsible for most of the scan time. (A similar scenario occurs when we lower the filtering fraction, accepting degraded sensitivity.)

Although the heuristic scans I performed use a practical amount of time, they do require

the use of compute clusters on the scale of work of this dissertation. Therefore, further improvements to speed will be helpful.

Independent filters. Multiple independent filters could be used if they complement each other. For example, although the ML-heuristic is generally superior to the BLAST-heuristic, it is possible that BLAST works better on some homologs. If so, combining information from both filters may improve overall filtering. I have not investigated whether these filters do complement each other. There may be limited improvements, since both filters are based on sequence information alone.

A more promising combination is the entirely sequence-based ML-heuristic and the entirely structure-based heuristic in FastR [4]. There is good reason to believe the two filters would complement each other since they use independent information.

Filter sensitivity. I recommend that the ML-heuristic be used at a filtering fraction of 0.01. However, some ncRNA families may be very easy (i.e., rely mostly on sequence information), so this filtering fraction would waste CPU time. For example, many families in Rfam contain highly similar sequences, leading to a very specific CM for which even BLAST filtering is fine.

To avoid wasting time, it would be helpful to be able to predict the sensitivity of the filter for a given CM. For easy families, it would be determined that a very low filtering fraction is sufficient. (Prediction of filter sensitivity would also be important if heuristic filters are to be run in series, adapting the methods of chapter 5 to the heuristic case.)

Two methods for sensitivity prediction present themselves. One is to test the filter on the known family members. This is risky, however, because it is unknown how diverged unknown homologs may be. Moreover, in many important cases, few homologs are known—often only one—so predictions would not be very robust. Another possibility is to use the method in chapter 9 to sample marginal hits directly from the CM. Although this sampling method was not effective in the context of discriminative filters, it may be adequate to predict sensitivity.

If the motivation for predicting filter sensitivity is to avoid wasting CPU time on easy

families, this motivation may be reduced as CMs are augmented with better priors. Better priors would make CMs less specific, and may lead to a situation where all families are equally hard.

Other information for filters. I have so far considered filtering based on sequence or secondary structure, but other information may be useful. Since most ncRNA families do not occur in protein-coding sequences, it may be reasonable to filter out predicted protein-coding genes. For prokaryotes, this could speed searches significantly.

Gene context can improve accuracy for *cis*-regulatory RNAs. For example, in the previous chapter, I found additional PIE RNAs—which are in the 3' UTRs of U1A spliceosomal proteins—in part by looking only downstream of U1A homologs. Protein-coding genes are easier to detect than ncRNAs, at least with current technologies, so restricting searches to regions around these genes can improve results. A disadvantage of this strategy, of course, is that RNA homologs that are not in the expected gene context may be missed.

Phylogenetic information may also be useful. For example, a large number of families in Rfam are confined to enterobacteria, a subset of γ-proteobacteria. It would significantly save time to search only the γ-proteobacteria for this family. If homologs are found in a significant subset of γ-proteobacteria, then the search could be repeated with the full eubacterial database (and if this is successful, with the full RFAMSEQ). If homologs are not found outside of enterobacteria, it is unlikely that they will be found outside of γ-proteobacteria, so the extra scanning time may be unwarranted.

A related scheme is to reduce redundancy in sequence databases, when the goal is to discover diverged homologs. For instance, if the goal is to try to discover an enterobacterial RNA within say the Firmicutes, it is not necessary to have 15 genomes of various strains of *Bacillus anthracis*. Note that this and the preceding schemes not only reduce search time, but also may improve overall sensitivity by reducing noise.

11.2 Applications

The main applications of the algorithms in this dissertation are: applying it to find homologs in the context of biology research, and creating family-specific filters. The first application

was discussed in the previous chapter. The creation of family-specific filters was implicitly described in the context of the SECIS element.

Other ncRNA families could use family-specific tools, and the ML-heuristic (and other techniques) provide a quick way to make the use of CMs practical. Although it is routine to create CMs (and therefore ML-heuristic filters) for any RNA family, family-specific tools can (1) incorporate additional types of information unique to the family, and (2) implement additional family-specific analysis.

As an example of the latter point, tRNAscan-SE automatically predicts the codon of putative tRNAs. A more sophisticated example is a pipeline that uses SECIS element prediction (among other information) to find selenoproteins.

There are many more families that may benefit from family-specific tools. With generic CMs and filters, development of these tools is much easier.

11.3 Improvements to ncRNA search algorithms

This dissertation has addressed a major limitation in CMs, namely their infeasibly slow scans. I believe that the major next step is to focus on other limitations of CMs. I will discuss these directions only briefly, since it is less related to my dissertation work.

11.3.1 Greater biological realism

Although CMs represent the most significant sources of information in ncRNAs—conservation of sequence and pseudoknot-free secondary structure—they miss pseudoknots, phylogeny and correlations between nucleotides other than base pairing. There have been a number of proposals for pseudoknots, but these are either an order of complexity more time consuming [21], or make other concessions, as with ERPIN [43].

For CMs, phylogeny has been considered for CMs in down-weighting training RNAs that are highly similar in sequence [32, 44]. However, more sophisticated uses are unknown. Phylogeny has been used to improve protein homology searches [101, 99, 117], and these methods often also model correlations between adjacent residues (e.g., in domains). Applying these solutions to RNA homology searches will be challenging due to the importance of

RNA secondary structure, and the increase in time complexity it may create.

There is also room for a better biological model in the local CM feature. Although there are clear biological motivations for this feature, their implementation seems to be strongly influenced by what features can easily and cheaply be accommodated by the CM Viterbi algorithm. Local begins and ends both require only modest changes in the algorithm and neither affects run time noticeably. Although this is very important, it may be possible to design something more biologically general, without inflating run time overly, or reducing specificity due to too much freedom in the model. Example directions would be (1) allowing cuts in single-stranded regions, and (2) perhaps only allowing a local begin or end at every alternate base pair, to reduce unnecessary freedom in the model.

A related direction is to compare current models in terms of accuracy on biological test sets: ERPIN, CMs and pattern-matching models like RNAmotif. Although there is some evidence to rate the models, it is not comprehensive—particularly for families that are not so well characterized. These comparisons may help to better understand the advantages and disadvantages of each scheme. It may be more difficult to compare pattern-matching models, since their performance is heavily dependent on the quality of pattern that an expert creates. By contrast, ERPIN's dependence on an expert for optimization seems like it would allow for less creativity, and therefore more objectivity. All three models depend on MSA quality (although pattern-matching techniques depend only indirectly).

11.3.2 Adopt family-specific features in generic framework

Family-specific tools have been created that exploit additional information, e.g., the existence of a homolog in a closely related organism, and matching snoRNAs to their reverse complement in target rRNAs. In principle, CMs could be extended to support both features in a general way. Although such extensions may not be easy, they would benefit subsequent research by making it easier for researchers to improve on existing tools, and by easing the development for tools on other families for which the features may be relevant.

11.3.3 Create MSAs automatically

Clearly the fact that CMs require an input MSA is inconvenient. This can be relieved by tools that automatically create MSAs of reasonable quality. In some cases, it may be helpful to reduce the need to tell the CM the secondary structure of an RNA. Especially in the cases of families with only one known member, current technology cannot plausibly predict structure well, e.g., single-sequence techniques like Mfold have poor accuracy [25].

Chapter 12

CONCLUSION

For decades, RNAs have appeared to be mere protein-coding messengers, with exceptions for tRNAs, rRNAs and spliceosomal RNAs. The last roughly five years has seen a torrent of discoveries showing that many more ncRNAs exist, that ncRNAs can take a dominant role in the cell and that presumably many more ncRNAs remain to be discovered.

Unfortunately, computational support for ncRNA research is behind that of protein-coding genes in terms of quality. In addition to the greater research focus on proteins, proteins may be easier, since they are longer and the larger alphabet size (20 amino acids) implies more information per position than RNA's nucleotides. Thus, for example, homology search for proteins can be reasonably successful using only the primary sequence information. For RNAs, this is generally inadequate; rather, RNA secondary structure is often crucial to accurate homology searches.

CMs have offered an elegant theory that yields excellent accuracy yet requires relatively little manual effort. However, CMs' exorbitant CPU requirements have limited their applicability. This dissertation has ameliorated this situation, speeding CMs enough to make them practical.

Using rigorous filters, this acceleration comes at no cost to sensitivity. This offers some important theoretical and practical advantages, and is an unusual achievement for a filter-based algorithm; generally filtering schemes in computational biology offer weaker guarantees or no guarantees.

At a faster speed, the ML-heuristic offers excellent sensitivity—comparable to the specialized tRNAscan-SE. Rigorous filters and the ML-heuristic represent a significant advance over previous ncRNA homology search tools.

Indeed, these tools have already been applied to current biology research, where they played a significant role in assisting research—discovering the cooperative-binding nature of

the glycine riboswitch and the widespread extent of 6S RNA. Moreover, the tool is available for other groups, so raises capabilities among RNA researchers.

To conclude, I evaluate the filters available and show the advantages of the ML-heuristic. I then discuss some higher-level issues on which my dissertation work has been instructive for me.

12.1 Evaluation of CM filters

In this section, I consider criteria for a successful CM filter, and then analyze various CM filter algorithms, including ones described in this dissertation. For a summary of conclusions, see Table 12.1.

To maximize computational contributions to ncRNA research, we need filters that are:

1. **Generic**, i.e., work for any ncRNA family. Given that Rfam has 503 families in it, there is a need for filters that can scale to support many families without manual effort.

2. **Sensitive**. (Note that—unlike sensitivity—specificity is not usually a concern, since the CM will eventually be run. Even if a filter eliminates nothing, the final specificity is that of the CM.)

3. **Fast**. Speed comes from two factors: (1) the speed of a scan with the filter (for example, the CM itself could be used as a filter with excellent sensitivity, but such a filter would obviously not speed homology searches), (2) the amount of sequence that a filter can eliminate—filters that eliminate more sequence will lead to faster overall homology searches.

4. **Able to trade sensitivity for speed**, e.g., if computer resources are limited. In general, heuristics have a tunable parameter that allows this tradeoff, but rigorous filters by definition cannot sacrifice sensitivity.

5. **Adaptable to improvements on CMs**—better CM technology should not invalidate the filters. Rigorous filters have problems on this point, since more complicated CM technology may make proofs of rigorousness hard to formulate.

   By contrast, although sophisticated heuristic filters may be hard to adapt to a more complicated CM, it should always be possible to create a non-improved CM, and use the resulting heuristic filter to filter searches with the improved CM. This solution is not ideal, since ideally the heuristic filter would be aware of the more sophisticated CM. However, the tactic yields a valid heuristic, and should have reasonable sensitivity for its given ncRNA. Indeed, the existing BLAST-heuristic entirely ignores the CM.

12.1.1 BLAST heuristic

There are theoretical reasons and empirical results suggesting that the BLAST-heuristic it not an ideal heuristic. The main theoretical problem is that BLAST is based purely on sequence information, yet RNA secondary structure is known to be very important to ncRNAs. By ignoring this, BLAST is at a significant handicap.

BLAST also requires short, exact word matches to speed its scans. When RNAs are conserved more on secondary structure, there may not be even a 7-nucleotide exact match. If the exact word match is eliminated, and an algorithm closer to Smith-Waterman [115, 27] is used, filter scans will be extremely time-consuming if several known ncRNAs are used in the query.

The empirical results of rigorous filters in chapter 4 and chapter 5 reveal that the BLAST heuristic does fail to find many ncRNAs. Chapter 7 empirically shows that other heuristics are more sensitive at any fixed speedup factor. This fact indicates that simply tuning BLAST's E-value cutoff would not be sufficient to allow it to be very accurate. BLAST's shortcomings are even more apparent in the local case of chapter 8.

In summary, BLAST's main disadvantage is its often poor sensitivity. As a heuristic filter, it has the advantage that it can trade sensitivity for speed, and that it is adaptable to improved CMs (albeit with possibly worse sensitivity). Among the generic heuristic filters, BLAST is the fastest.

Table 12.1: Comparison of sequence filters for CMs. Various sequence filters are compared in terms of multiple criteria. The second column represents both tRNAscan-SE itself (which is not applicable to other ncRNA families), and a family-specific strategy similar to tRNAscan-SE. The main advantage of this strategy is superior performance, assuming the results of tRNAscan-SE extrapolate to other ncRNA families. Unfortunately, this strategy's cost is a potentially large investment for each new ncRNA family. The next column is the rigorous filters described in this dissertation. The main advantage is guaranteed perfect sensitivity; the disadvantage is the often slow speed, and inability to trade sensitivity for speed. BLAST refers to the use of BLAST in the Rfam Database. The main advantage is that BLAST is generic to any ncRNA family, and quite fast. Unfortunately, sensitivity is often problematic. ML-heur is the ML-heuristic described in this dissertation. The main advantages are solid sensitivity and speed for a completely generic method. There is no one disadvantage, but clearly there is room to improve sensitivity and speed even further.

	strategy like tRNAscan-SE	**rigorous**	**BLAST**	**ML-heur**
generic	No. Significant work to extend to new family	Yes	Yes	Yes
Sensitivity	Excellent	Perfect	Often poor	Excellent
Speed	Excellent	Often slow	Excellent	Very good
Can trade sensitivity for speed	Yes	No	Yes	Yes
Adaptable to improved CMs	Yes	Unclear	Yes	Yes

12.1.2 tRNAscan-SE and family-specific filters

tRNAscan-SE inherits the excellent sensitivity and specificity of the CMs it uses. (Due to the filters, specificity is improved, and sensitivity slightly degraded relative to a CM.) The filters in tRNAscan-SE make it relatively fast, compared to other statistical profile tools.

The main disadvantage is that tRNAscan-SE cannot be used for other families, as noted in chapter 2 and chapter 7. Thus, tRNAscan-SE is not a general-purpose solution for ncRNA homology search. Moreover, a family-specific strategy like tRNAscan-SE's cannot easily be adopted to other families, since it will require creating non-CM detection tools for those families.

12.1.3 Rigorous filters

Rigorous filters guarantee perfect sensitivity, which eliminates the need to design better heuristic filters—we know that nothing can be better.

However, rigorous filters fail on some of the criteria. First, they are slower than heuristics can be, because rigorous guarantees are required.

Moreover, rigorous filters cannot trade sensitivity for increased speed, again since they must guarantee perfect sensitivity. For example, rigorous scans of 1.7 Gbases of bacterial sequences for glycine riboswitch homologs [86], took 9.7 CPU days, while the ML-heuristic took 1.8 days, yet missed only 1 putative homolog (out of 609). The rigorous filtering techniques could not run faster without sacrificing rigorousness.

Next, rigorous filters failed to improve scanning time for two families I have tested, i.e., it could not speed CMs significantly while maintaining guarantees. Although two families (out of 139 tested) is a modest problem, it would be helpful to search these families in a practical amount of time. Moreover, as ncRNA research progresses, I anticipate that more known families will become bigger, and their secondary structure thus more prominent. So, more highly structured families may emerge that challenge the rigorous filters described in this dissertation, which are best at exploiting primary sequence conservation.

Finally, the proofs used in rigorous filters may not translate to improvements on CMs.

12.1.4 ML-heuristic

As a heuristic filter, the ML-heuristic has the advantage that it can trade sensitivity for speed, and that it is adaptable to improved CMs (albeit with possibly worse sensitivity). Like BLAST, it is generic, and can be automatically applied to any family. (Creating an ML-heuristic filter for a new ncRNA family, given its CM, takes roughly 15 minutes on a 2.8 GHz Pentium IV, and requires no manual effort other than entering a command line.)

Although the ML-heuristic leads to slower scans than the BLAST-heuristic, the difference in speed is not drastic (roughly a factor of 2-4). Most significantly, the ML-heuristic is often much more sensitive than the BLAST-heuristic. Looking for extremely diverged homologs (as in chapter 8), the BLAST-heuristic is often unable to find any homologs at all, where the ML-heuristic finds several.

The ML-heuristic proves comparable to (though not as good as) tRNAscan-SE. Sensitivity and specificity are very similar. While the ML-heuristic is slower (by 3-12 times), its run time is practical. It is very encouraging that this generic filter is comparable to a highly specialized tool like tRNAscan-SE.

The ML-heuristic addresses all the serious shortcomings of other filters: the family-specific nature of tRNAscan-SE, the inflexible slowness of rigorous filters (for some families) and the often poor sensitivity of BLAST. Future work may improve further on its speed and sensitivity.

12.2 Higher-level conclusions

12.2.1 Impractically slow algorithms can be useful

Without filters, CMs require infeasible amounts of computer time. When they were first developed in 1994, significantly fewer RNA families were known, although CPUs were also considerably slower than now. Overall, a hypothetical 1994 version of Rfam would have been, like today, infeasible with a pure CM solution. CMs were only practical running one family (most likely tRNA) on very small sequences.

However, CMs had three advantages: excellent accuracy, more modest manual effort and an elegant theoretical basis making further development easier. Despite their infeasible

time requirements, algorithms were developed to make CMs faster with modest degradation of their advantages: first with tRNAscan-SE in 1997, and now for all families.

This history demonstrates that elegantly "nice" algorithms can be useful even if they are infeasibly slow. Thus, as computational biologists, we should be happy to develop impractically slow algorithms that make important advances—in 10 years, someone may improve the speed.

12.2.2 Rigorous vs. heuristic

It is common in computational biology and other fields of computer science to run into nice algorithms that require too much time. Proposed solutions usually involve heuristics, and rigorousness is seldom achieved.

There are two types of rigorous solutions of which I am aware:

1. **100% rigorous.** As in this dissertation, 100% rigorous techniques provably guarantee that they sacrifice none of the slower algorithms sensitivity.

2. **probabilistically rigorous.** These algorithms prove that the probability of missing some result (like an RNA homolog) is p, and usually with more CPU time it is possible to make p arbitrarily close to 0. An example of this work is Jeremy Buhler's alternate BLAST-like heuristics [15]. (Note: if $p = 0$, then probabilistic rigor becomes the 100% rigor.)

From a purely practical perspective, rigorous solutions provide important advantages. They make analysis of the algorithm's sensitivity easy. They also shut down speculation on whether an improved heuristic is needed to find more results. If I had developed the ML-heuristic first, I would have found that it did not add many homologs for most families, and would have had to decide whether to try to improve the heuristic. As it turns out, based on the results of my rigorous filters, I know that for those Rfam families, BLAST is sufficient and there is nothing more to be found. (Or CMs need improvements such as better priors.)

Disadvantages of a rigorous approach, however, are also significant. First, it may be too difficult to design such techniques. Second, this problem can become arbitrarily harder as the CM (or other "nice" algorithms) becomes more sophisticated. At a certain point, it will probably not pay to make more and more tricky proofs of rigorousness. Finally, as in my case, such solutions may sacrifice speed or other features.

Overall, for most problems rigorous filters are useful at some point, but heuristics may be the most practical solution in many cases. Rigorous filters can be used to validate heuristics, at which point increased confidence in heuristics lessens the need for rigorous filters.

12.2.3 The application developer as applier

Although my main contribution has been to develop algorithms for ncRNA homology search, I also applied the work to biological problems. I both assisted experimental research in collaboration with the Breaker Lab, and iteratively improved MSAs.

This work has had a number of benefits. First, the developer of algorithms and software is generally the expert on their use, and therefore the most capable user early on. This expert use can spur biological developments using the software.

Second, having some experience in the application domain can help to improve one's understanding of what is important, and suggest other avenues for research. For example, experiences where CMs failed suggested that implementing filters for local CMs would be useful. Learning overall problems faced in the lab suggested other open research problems directly and indirectly related to the actual dissertation work.

Third, early success with the software motivates others to use it. If the algorithm developer publishes papers, and lets others apply the software, those others may have more difficulty applying the software or may not be convinced the effort is worthwhile. By showing that the software can generate important biological results, others can see that it is worth their time to learn the software.

BIBLIOGRAPHY

[1] Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[2] Victor Ambros, Bonnie Bartel, David P. Bartel, Christopher B. Burge, James C. Carrington, Xuemei Chenand, Gideon Dreyfuss, Sean R. Eddy, Sam Griffiths-Jones, Mhairi Marshall, Marjori Matzke, Gary Ruvkun, and Thomas Tuschl. A uniform system for microRNA annotation. *RNA*, 9:277–279, 2003.

[3] Liron Argaman, Ruth Hershberg, Jörg Vogel, Gill Bejerano, E. Gerhart H. Wagner, Hanah Margalit, and Shoshy Altuvia. Novel small RNA-encoding genes in the intergenic regions of *Escherichia coli*. *Current Biology*, 11(12):941–950, 2001.

[4] Vineet Bafna and Shaojie Zhang. FastR: Fast database search tool for non-coding RNA. In *IEEE Computational Systems Bioinformatics Conference 2004*, pages 52–61. IEEE, 2004.

[5] Jeffrey E. Barrick, Keith A. Corbino, Wade C. Winkler, Ali Nahvi, Maumita Mandal, Jennifer Collins, Mark Lee, Adam Roth, Narasimhan Sudarsan, Inbal Jona, J. Kenneth Wickiser, and Ronald R. Breaker. New RNA motifs suggest an expanded scope for riboswitches in bacterial genetic control. *Proc Natl Acad Sci USA*, 101(17):6421–26, 2004.

[6] Jeffrey E. Barrick, N. Sudarsan, Zasha Weinberg, Walter L. Ruzzo, and Ronald R. Breaker. 6S RNA is a widespread regulator of eubacterial RNA polymerase that resembles an open promoter. *RNA*, 11:774–84, 2005.

[7] Alex Bateman, Lachlan Coin, Richard Durbin, Robert D. Finn, Volker Hollich, Sam Griffiths-Jones, Ajay Khanna, Mhairi Marshall, Simon Moxon, Erik L. L. Sonnhammer, David J. Studholme, Corin Yeats, and Sean R. Eddy. The Pfam protein families database. *Nucleic Acids Research*, 32:D138–D141, 2004.

[8] Fabia U Battistuzzi, Andreia Feijao, and S Blair Hedges. A genomic timescale of prokaryote evolution: insights into the origin of methanogenesis, phototrophy, and the colonization of land. *BMC Evo Biol*, 4:44, 2004.

[9] D. Baulcombe. An RNA microcosm. *Science*, 297(5589):2002–2003, 2002.

[10] Isaac Bentwich, Amir Avniel, Yael Karov, Ranit Aharonov, Shlomit Gilad, Omer Barad, Adi Barzilai, Paz Einat, Uri Einav, Eti Meiri, Eilon Sharon, Yael Spector, and Zvi Bentwich. Identification of hundreds of conserved and nonconserved human microRNAs. *Nature Genetics*, 37(7):766–70, 2005.

[11] S. Bonhoeffer, J.S. McCaskill, P.F. Stadler, and P. Schuster. RNA multistructure landscapes - a study based on temperature-dependent partition-functions. *European Biophysics Journal with Biophysics Letters*, 22(1):13–24, 1993.

[12] Sabine Brantl. Antisense-RNA regulation and RNA interference. *Biochimica et Biophysica Acta*, 1575:15–25, 2002.

[13] Sabine Brantl. Antisense RNAs in plasmids: control of replication and maintenance. *Plasmid*, 48:165–173, 2002.

[14] M.A. Brinton, A.V. Fernandez, and J.H. Dispoto. 3'-nucleotides of flavivirus genomic RNA form a conserved secondary structure. *Virology*, 153(1):113–121, 1986.

[15] Jeremy Buhler. *Search Algorithms for Biosequences Using Random Projection.* PhD thesis, University of Washington, 2001.

[16] Ashley H. Carter. *Classical and statistical thermodynamics.* Prentice Hall, Upper Saddle River, NJ, USA, 2001.

[17] Richard J. Carter, Inna Dubchak, and Stephen R. Holbrook. A computational approach to identify genes for functional RNAs in genomic sequences. *Nucleic Acids Research*, 29(19):3928–3938, 2001.

[18] Sergi Castellano, Sergey V. Novoselov, Gregory V. Kryukov, Alain Lescure, Enrique Blanco, Alain Krol, Vadim N. Gladyshev, and Roderic Guigó. Reconsidering the evolution of eukaryotic selenoproteins: a novel nonmammalian family with scattered phylogenetic distribution. *EMBO Rep.*, 5(1):71–77, 2004.

[19] Nathalie Charlier, Pieter Leyssen, Cornelis W. A. Pleij, Philippe Lemey, Frédérique Billoir, Kristel Van Laethem, Anne-Mieke Vandamme, Erik De Clercq, Xavier de Lamballerie, and Johan Neyts. Complete genome sequence of Montana *Myotis* leukoencephalitis virus, phylogenetic analysis and comparative study of the 3' untranslated region of flaviviruses with no known vector. *Journal of General Virology*, 83:1875–1885, 2002.

[20] Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124, 1956.

[21] Anne Condon, Beth Davy, Baharak Rastegari, Finbarr Tarrant, and Shelly Zhao. Classifying RNA pseudoknotted structures. *Theoretical Computer Science*, 320(1):35–50, 2004.

[22] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms.* MIT Press, Cambridge, USA, 1999.

[23] M.B. Crabtree, R. C. Sang, V. Stollar, L. M. Dunster, and B. R. Miller. Genetic and phenotypic characterization of the newly described insect flavivirus, Kamiti River virus. *Archives of Virology*, 148:1095–1118, 2003.

[24] Decatur and Fournier. RNA-guided nucleotide modification of ribosomal and other RNAs. *Journal of Biological Chemistry*, 278(2):695–698, 2003.

[25] Robin D. Dowell and Sean R. Eddy. Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. *BMC Bioinformatics*, 5:71, 2004.

[26] M. Dsouza, N. Larsen, and R. Overbeek. Searching for patterns in genomic data. *Trends in Genetics*, 13(12):497–498, 1997.

[27] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis.* Cambridge University Press, Cambridge, UK, 1998.

[28] Sean R. Eddy. *Maximum Likelihood Fitting of Extreme Value Distributions*, 1997. `http://selab.wustl.edu/publications/Eddy97b/Eddy97b-techreport.pdf`.

[29] Sean R. Eddy. Profile hidden Markov models. *Bioinformatics*, 14:755–763, 1998.

[30] Sean R. Eddy. Computational genomics of noncoding RNA genes. *Cell*, 109:137–140, 2002.

[31] Sean R. Eddy. A memory-efficient dynamic programming algorithm for optimal alignment of a sequence to an RNA secondary structure. *BMC Bioinformatics*, 3:18, 2002.

[32] Sean R. Eddy. *Infernal User's Guide*, 2003. `ftp://ftp.genetics.wustl.edu/pub/eddy/software/infernal/Userguide.pdf`.

[33] Sean R. Eddy and R. Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Research*, 22(11):2079–2088, 1994.

[34] Sean R. Eddy, Graeme Mitchison, and Richard Durbin. Maximimum discrimination hidden Markov models of sequence consensus. *J Comp Bio*, 2:9–23, 1995.

[35] Sverker Edvardsson, Paul P. Gardner, Anthony M. Poole, Michael D. Hendy, David Penny, and Vincent Moulton. A search for H/ACA snoRNAs in yeast using MFE secondary structure prediction. *Bioinformatics*, 19(7):865–873, 2003.

[36] Volker A. Erdmann, Miroslawa Z. Barciszewska, Mariej Szymanski, Abraham Hochberg, Nathan de Groot, and Jan Barciszewski. The non-coding RNAs as riboregulators. *Nucleic Acids Research*, 29(1):189–193, 2001.

[37] Gerardo Ferbeyre, Véronique Bourdeau, Marie Pageau, Pedro Miramontes, and Robert Cedergren. Distribution of hammerhead and hammerhead-like RNA motifs through the GenBank. *Genome Research*, 10(7):1011–1019, 2000.

[38] Gwennaele A. Fichant and Christian Burks. Identifying potential tRNA genes in genomic DNA sequences. *Journal of Molecular Biology*, 220(3):659–671, 1991.

[39] Gary B. Fogel, V. William Porto, Dana G. Weekes, David B. Fogel, Richard H. Griffey, John A. McNeil, Elena Lesnik, David J. Ecker, and Rangarajan Sampath. Discovery of RNA structural elements using evolutionary computation. *Nucleic Acids Research*, 30(23):5310–5317, 2002.

[40] Howard E. Ganther. Selenium metabolism, selenoproteins and mechanisms of cancer prevention: complexities with thioredoxin reductase. *Carcinogenesis*, 20(9):1657–1666, 1999.

[41] Paul P Gardner and Robert Giegerich. A comprehensive comparison of comparative RNA structure prediction approaches. *BMC Bioinformatics*, 5:140, 2004.

[42] Paul P. Gardner, Andreas Wilm, and Stefan Washietl. A benchmark of multiple sequence alignment programs upon structural RNAs. *Nucleic Acids Research*, 33:2433–2439, 2005.

[43] Daniel Gautheret and André Lambert. Direct RNA motif definition and identification from multiple sequence alignments using secondary structure profiles. *Journal of Molecular Biology*, 313:1003–1011, 2001.

[44] Mark Gerstein, Erik L. L. Sonnhammer, and Cyrus Chothia. Volume changes in protein evolution. *J Mol Biol*, 236(4):1067–1078, 1994.

[45] Susan Gottesman. Stealth regulation: biological circuits with small RNA switches. *Genes and Development*, 16:2829–2842, 2002.

[46] Anthony J.F. Griffiths, Jeffrey H. Miller, David T. Suzuki, Richard C. Lewontin, and William M. Gelbart. *An introduction to genetic analysis.* W.H. Freeman, New York, USA, 2000.

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

[48] Sam Griffiths-Jones, Simon Moxon, Mhairi Marshall, Ajay Khanna, Sean R. Eddy, and Alex Bateman. Rfam: annotating non-coding RNAs in complete genomes. *Nucleic Acids Research*, 33:D121–D124, 2005.

[49] Giorgio Grillo, Flavio Licciulli, Sabino Liuni, Elisabetta Sbisà, and Graziano Pesole. PatSearch: a program for the detection of patterns and structural motifs in nucleotide sequences. *Nucleic Acids Research*, 31(13):3608–3612, 2003.

[50] Frank J. Grundy, Sean M. Rollins, and Tina M. Henkin. Interaction between the acceptor end of tRNA and the T box stimulates antitermination in the *Bacillus subtilis* tyrS gene: a new role for the discriminator base. *J Bacteriol 1994*, 176:4518–26, 1994.

[51] Daniel H. Haft, Jeremy D. Selengut, and Owen White. The TIGRFAMs database of protein families. *Nucleic Acids Research*, 31:371–373, 2003.

[52] Lin He and Gregory J. Hannon. microRNAs: small RNAs with a big role in gene regulation. *Nat Rev Genet*, 5(7):522–531, 2004.

[53] I. Holmes and G.M. Rubin. Pairwise RNA structure comparison with stochastic context-free grammars. *Pacific Symposium Biocomputing*, pages 163–174, 2002.

[54] Fan Hsu, Tom H. Pringle, Robert M. Kuhn, Donna Karolchik, Mark Diekhans, David Haussler, and W. James Kent. The UCSC proteome browser. *Nucleic Acids Research*, 33:D454–D458, 2005.

[55] Alexander Hüttenhofer, Jürgen Brosius, and Jean-Pierre Bachellerie. RNomics: identification and function of small, non-messenger RNAs. *Current Opinion in Chemical Biology*, 6:835843, 2002.

[56] Alexander Hüttenhofer, Peter Schattner, and Norbert Polacek. Non-coding RNAs: hope or hype? *Trends in Genetics*, 21:289–297, 2005.

[57] Tommi Jaakkola, Mark Diekhans, and David Haussler. A discriminative framework for detecting remote protein homologies. *J Comp Bio*, 7(1):95–114, 2000.

[58] Arun Jagota, Rune B. Lyngsø, and Christian N.S. Pedersen. Comparing an HMM and an SCFG. *Proceedings of the 1st Workshop on Algorithms in Bioinformatics (WABI 01)*, pages 69–84, 2001.

[59] Biing-Hwang Juang and Shigeru Katagiri. Discriminative learning for minimum error classification. *IEEE Trans Sig Proc*, 40(12):3043–3054, 1992.

[60] Donald Kennedy. Breakthrough of the year. *Science*, 298(5602):2283, 2002.

[61] Robbie J. Klein and Sean R. Eddy. RSEARCH: finding homologs of single structured RNA sequences. *BMC Bioinformatics*, 4(1):44, 2003.

[62] Robert J. Klein, Ziva Misulovin, and Sean R. Eddy. Noncoding RNA genes identified in AT-rich hyperthermophiles. *Proceedings of the National Academy of Sciences of the USA*, 99(11):7542–7547, 2002.

[63] Peter S. Klosterman, Makio Tamura, Stephen R. Holbrook, and Steven E. Brenner. SCOR: a structural classification of RNA database. *Nucleic Acids Research*, 30(2):392–394, 2002.

[64] Cas Kramer, Jennifer J. Loros, Jay C. Dunlap, and Susan K. Crosthwaite. Role for antisense RNA in regulating circadian clock function in *Neurospora crassa*. *Nature*, 421:948–952, 2003.

[65] Anders Krogh and Graeme Mitchison. Maximum entropy weighting of aligned sequence of proteins or DNA. In *Proc. 3rd Inter. Conf. on Intelligent Systems for Molecular Biology*, pages 215–221. AAAI Press, 1995.

[66] Alain Krol. Evolutionarily different RNA motifs and RNA-protein complexes to achieve selenoprotein synthesis. *Biochimie*, 84(8):765–774, 2002.

[67] Gregory V. Kryukov, Sergi Castellano, Sergey V. Novoselov, Alexey V. Lobanov, Omid Zehtab, Roderic Guigó, and Vadim N. Gladyshev. Characterization of mammalian selenoproteomes. *Science*, 300(5624):1439–1443, 2003.

[68] Gregory V. Kryukov and Vadim N. Gladyshev. The prokaryotic selenoproteome. *EMBO Rep.*, 5(5):538–543, 2004.

[69] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Inter. Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, 2001.

[70] Mariana Lagos-Quintana, Reinhard Rauhut, Winfried Lendeckel, and Thomas Tuschl. Identification of novel genes coding for small expressed RNAs. *Science*, 294(5543):853–858, 2001.

[71] Eric C. Lai. Micro RNAs are complementary to 3' UTR sequence motifs that mediate negative post-transcriptional regulation. *Nature Genetics*, 30:363–364, 2002.

[72] Eric C. Lai. RNA sensors and riboswitches: Self-regulating messages. *Current Biology*, 13:R285–R291, 2003.

[73] André Lambert, Jean-Fred Fontaine, Matthieu Legendre, Fabrice Leclerc, Emmanuelle Permal, François Major, Harald Putzer, Olivier Delfour, Bernard Michot, and Daniel Gautheret. The ERPIN server: an interface to profile-based RNA motif identification. *Nucleic Acids Research*, 32:W160–W165, 2004.

[74] André Lambert, Alain Lescure, and Daniel Gautheret. A survey of metazoan selenocysteine insertion sequences. *Biochimie*, 84:953–959, 2002.

[75] Dean Laslett and Bjorn Canback. ARAGORN, a program to detect tRNA genes and tmRNA genes in nucleotide sequences. *Nucleic Acids Research*, 32:11–16, 2004.

[76] Dean Laslett, Bjorn Canback, and Siv Andersson. BRUCE: a program for the detection of transfer-messenger RNA genes in nucleotide sequences. *Nucleic Acids Research*, 30:3449–3453, 2002.

[77] Craig Lawrence, Jian L. Zhou, and André L. Tits. User's guide for CFSQP version 2.5: A C code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints. Technical report, Institute for Systems Research, University of Maryland, College Park, 1997. TR-94-16r1.

[78] Shu-Yun Le, Kaizhong Zhang, and Jacob V. Maizel Jr. RNA molecules with structure dependent functions are uniquely folded. *Nucleic Acids Research*, 30(16):3574–3582, 2002.

[79] Elena A. Lesnik, Ranga Sampath, and David J. Ecker. Rev response elements (RRE) in lentiviruses: an RNAMotif algorithm-based strategy for RRE prediction. *Medicinal Research Reviews*, 22(6):617–636, 2002.

[80] Lee P. Lim, Margaret E. Glasner, Soraya Yekta, Christopher B. Burge, and David P. Bartel. Vertebrate MicroRNA genes. *Science*, 299(5612):1540, 2003.

[81] Lee P. Lim, Nelson C. Lau, Earl G. Weinstein, Aliaa Abdelhakim, Soraya Yekta, Matthew W. Rhoades, Christopher B. Burge, and David P. Bartel. The microRNAs of *Caenorhabditis elegans*. *Genes and Development*, 17(8):991–1008, 2003.

[82] Todd M. Lowe and Sean R. Eddy. tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence. *Nucleic Acids Research*, 25(5):955–64, 1997.

[83] Todd M. Lowe and Sean R. Eddy. A computational screen for methylation guide snoRNAs in yeast. *Science*, 283:1168–1171, 1999.

[84] Thomas J. Macke, David J. Ecker, Robin R. Gutell, Daniel Gautheret, David A. Case, and Rangarajan Sampath. RNAMotif, an RNA secondary structure definition and search algorithm. *Nucleic Acids Research*, 29(22):4724–4735, 2001.

[85] Matilde Maiorino, Valentina Boselloa, Fulvio Ursinia, Carlo Forestab, Andrea Garollab, Margherita Scapina, Helena Sztajerc, and Leopold Flohéc. Genetic variations of gpx-4 and male infertility in humans. *Biol Reprod*, 68(4):1134–1141, 2003.

[86] Maumita Mandal, Mark Lee, Jeffrey E. Barrick, Zasha Weinberg, Gail Mitchell Emilsson, Walter L. Ruzzo, and Ronald R. Breaker. A glycine-dependent riboswitch that uses cooperative binding to control gene expression in bacteria. *Science*, 306:275–279, 2004.

[87] Claudia Marker, Anja Zemann, Tanja Terhörst, Martin Kiefmann, James P. Kastenmayer, Pamela Green, Jean-Pierre Bachellerie, Jürgen Brosius, and Alexander Hüttenhofer. Experimental RNomics: identification fo 140 candidates for small non-messenger RNAs in the plant *Arabidopsis thaliana*. *Current Biology*, 12:2002–2013, 2002.

[88] John S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structures. *Biopolymers*, 29:1105–1119, 1990.

[89] John P. McCutcheon and Sean R. Eddy. Computational identification of non-coding RNAs in *Saccharomyces cerevisiae* by comparative genomics. *Nucleic Acids Research*, 31:4119–4128, 2003.

[90] Charles E. Metz. Basic principles of ROC analysis. *Semin Nucl Med*, 8(4):283–298, 1978.

[91] Mehryar Mohri and Mark-Jan Nederhof. Regular approximation of context-free grammars through transformation. In Jean-Claude Junqua and Gertjan van Noord, editors, *Robustness in language and speech technology*, pages 251–261. Kluwer Academic Publishers, 2000.

[92] Peter B. Moore. Structural motifs in RNA. *Annual Reviews in Biochemistry*, 68:287–300, 1999.

[93] Eric G. Moss. MicroRNAs: hidden in the genome. *Current Biology*, 12(4):R138–140, 2002.

[94] Vincent Moulton, Michael Zuker, Michael Steel, Robin Pointon, and David Penny. Metrics on RNA secondary structures. *Journal of Computational Biology*, 7(1-2):277–292, 2000.

[95] Nicola J. Mulder, Rolf Apweiler, Teresa K. Attwood, Amos Bairoch, Alex Bateman, David Binns, Paul Bradley, Peer Bork, Phillip Bucher, Lorenzo Cerutti, Richard Copley, Emmanuel Courcelle, Ujjwal Das, Richard Durbin, Wolfgang Fleischmann, Julian Gough, Daniel Haft, Nicola Harte, Nicolas Hulo, Daniel Kahn, Alexander Kanapin, Maria Krestyaninova, David Lonsdale, Rodrigo Lopez, Ivica Letunic, Martin Madera1, John Maslen, Jennifer McDowall, Alex Mitchell, Anastasia N. Nikolskaya, Sandra Orchard, Marco Pagni, Chris P. Ponting, Emmanuel Quevillon, Jeremy Selengut, Christian J. A. Sigrist, Ville Silventoinen, David J. Studholme, Robert Vaughan, and Cathy H. Wu. InterPro, progress and status in 2005. *Nucleic Acids Research*, 33:D201–D205, 2005.

[96] A. Pavesi, F. Conterio, A. Bolchi, G. Dieci, and S. Ottonello. Identification of new eukaryotic tRNA genes in genomic DNA databases by a multistep weight matrix analysis of transcriptional control regions. *Nucleic Acids Research*, 22(7):1247–1256, 1994.

[97] Brenda A. Peculis. RNA-binding proteins: if it looks like a sn(o)RNA... *Current Biology*, 10(24):R916–R918, 2000.

[98] Kim D. Pruitt, Tatiana Tatusova, and Donna R. Maglott. NCBI reference sequence (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Research*, 33:D501–D504, 2005.

[99] Bin Qian and Richard A. Goldstein. Detecting distant homologs using phylogenetic tree-based HMMs. *Proteins*, 52:446–453, 2003.

[100] Marco Regalia, Magnus Alm Rosenblad, and Tore Samuelsson. Prediction of signal recognition particle RNA genes. *Nucleic Acids Research*, 30:3368–3377, 2002.

[101] Marc Rehmsmeier and Martin Vingron. Phylogenetic information improves homology detection. *Proteins*, 45:360–371, 2001.

[102] Brenda J. Reinhart, Earl G. Weinstein, Matthew W. Rhoades, Bonnie Bartel, , and David P. Bartel. MicroRNAs in plants. *Genes and Development*, 16(13):1616–1626, 2002.

[103] Matthew W. Rhoades, Brenda J. Reinhart, Lee P. Lim, Christopher B. Burge, Bonnie Bartel, and David P. Bartel. Prediction of plant microRNA targets. *Cell*, 110:513–520, 2002.

[104] Elena Rivas and Sean R. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of Molecular Biology*, 285:2053–2068, 1999.

[105] Elena Rivas and Sean R. Eddy. The language of RNA: a formal grammar that includes pseudoknots. *Bioinformatics*, 16(4):334–340, 2000.

[106] Elena Rivas and Sean R. Eddy. Secondary structure alone is generally not statistically significant for the detection of noncoding RNAs. *Bioinformatics*, 16(7):583–605, 2000.

[107] Elena Rivas and Sean R. Eddy. Noncoding RNA gene detection using comparative sequence analysis. *BMC Bioinformatics*, 2(1):8, 2001.

[108] Elena Rivas, Robert J. Klein, Thomas A. Jones, and Sean R. Eddy. Computational identification of noncoding RNAs in *E. coli* by comparative genomics. *Current Biology*, 11:1369–1373, 2001.

[109] Stuart J. Russell and Peter Norvig. *Artificial intelligence: a modern approach.* Prentice Hall, Upper Saddle River, NJ, USA, 1995.

[110] Yasubumi Sakakibara, Michael Brown, Richard Hughey, I Saira Mian, Kimmen Sjölander, Rebecca C Underwood, and David Haussler. Stochastic context-free grammars for tRNA modeling. *Nuc Acids Research*, 22(23):5112–5120, 1994.

[111] Peter Schattner. Searching for RNA genes using base-composition statistics. *Nucleic Acids Research*, 30(9):2076–2082, 2002.

[112] Ulrich Schweizer, Lutz Schomburg, and Nicolai E. Savaskan. The neurobiology of selenium: Lessons from transgenic mice. *J Nutr*, 134:707–710, 2004.

[113] Pei-Yong Shi, Margo A. Brinton, James M. Veal, Yi Yi Zhong, and W. David Wilson. Evidence for the existence of a pseudoknot structure at the 3' terminus of the flavivirus genomic RNA. *Biochemistry*, 35(13):4222–4230, 1996.

[114] Joanna L. Shisler, Tatiana G. Senkevich, Marla J. Berry, and Bernard Moss. Ultraviolet-induced cell death blocked by a selenoprotein from a human dermatotropic poxvirus. *Science*, 279:102–105, 1998.

[115] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[116] George W. Snedecor and William G. Cochran. *Statistical methods.* Iowa State University Press, Ames, Iowa, USA, eighth edition, 1989.

[117] Rainer Spang, Marc Rehmsmeier, and Jens Stoye. A novel approach to remote homology detection: jumping alignments. *J Comput Biol*, 9(5):747–760, 2002.

[118] Guenter Stoesser, Wendy Baker, Alexandra van den Broek, Maria Garcia-Pastor, Carola Kanz, Tamara Kulikova, Rasko Leinonen, Quan Lin, Vincent Lombard, Rodrigo Lopez, Renato Mancuso, Francesco Nardone, Peter Stoehr, Mary Ann Tuli, Katerina Tzouvara, and Robert Vaughan. The EMBL nucleotide sequence database: major new developments. *Nucleic Acids Research*, 31(1):17–22, 2003.

[119] Gisela Storz. An expanding universe of noncoding RNAs. *Science*, 296(5571):1260–1263, 2002.

[120] N. Sudarsan, Jeffrey E. Barrick, and Ronald R. Breaker. Metabolite-binding RNA domains are present in the genes of eukaryotes. *RNA*, 9:644–647, 2003.

[121] Philip R. Taylor, Howard L. Parnes, and Scott M. Lippman. Science peels the onion of selenium effects on prostate carcinogenesis. *J Natl Cancer Inst*, 96(9):645–647, 2004.

[122] C. D. Thomson. Assessment of requirements for selenium and adequacy of selenium status: a review. *Eu J Clin Nutr*, 58(3):391–402, 2004.

[123] Amy E. Trotochaud and Karen M. Wassarman. A highly conserved 6S RNA structure is required for regulation of transcription. *Nature Structural and Molecular Biology*, 12:313–319, 2005.

[124] Vickie Tsui, Tom Macke, and David A. Case. A novel method for finding tRNA genes. *RNA*, 9:507–517, 2003.

[125] Alexey G. Vitreschak, Dimitry A. Rodionov, Andrey A. Mironov, and Mikhail S. Gelfand. Riboswitches: the oldest mechanism for the regulation of gene expression? *Trends Genet.*, 20(1):44–50, 2004.

[126] E.G. Wagner and K. Flardh. Antisense RNAs everywhere? *TRENDS in Genetics*, 18(5):223–226, 2002.

[127] Jason Tsong-Li Wang and Kaizhong Zhang. Identifying consensus of trees through alignment. *Information Sciences*, 126:165–189, 2000.

[128] Karen Wassarman, F. Repoila, Carsten Rosenow, Gisela Storz, and Susan Gottesman. Identification of novel small RNAs using comparative genomics and microarrays. *Genes and Development*, 15(13):1637–1651, 2001.

[129] Zasha Weinberg. RAVENNA *software package.* `http://bio.cs.washington.edu/supplements/zasha-ravenna`.

[130] Zasha Weinberg and Walter L. Ruzzo. Faster genome annotation of non-coding RNA families without loss of accuracy. In *Proc. Eighth Annual Inter. Conf. on Computational Molecular Biology (RECOMB)*, pages 243–251. ACM Press, 2004.

[131] 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–i340, 2004.

[132] Eric W. Weisstein. *Extreme Value Distribution.* From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/ExtremeValueDistribution.html`.

[133] Dagmar K. Willkomm, Jens Minnerup, Alexander Hüttenhofer, and Roland K. Hartmann. Experimental RNomics in *Aquifex aeolicus*: identification of small non-coding RNAs and the putative 6S RNA homolog. *Nucleic Acids Research*, 33(6):1949–1960, 2005.

[134] Wade Winkler, Ali Nahvi, and Ronald R. Breaker. Thiamine derivatives bind messenger RNAs directly to regulate bacterial gene expression. *Nature*, 419:952–956, 2002.

[135] Wade C. Winkler and Ronald R. Breaker. Genetic control by metabolite-binding riboswitches. *Chembiochem.*, 4(10):1024–1032, 2003.

[136] Wade C. Winkler, Frank J. Grundy, Brooke A. Murphy, and Tina M. Henkin. The GA motif: An RNA element common to bacterial antitermination systems, rRNA, and eukaryotic RNAs. *RNA*, 7:116572, 2001.

[137] Wade C. Winkler, Ali Nahvi, Adam Roth, Jennifer A. Collins, and Ronald R. Breaker. Control of gene expression by a natural metabolite-responsive ribozyme. *Nature*, 428:281–286, 2004.

[138] Lingling Zeng, Barry Falgout, and Lewis Markoff. Identification of specific nucleotide sequences within the conserved 3'-SL in the Dengue type 2 virus genome required for replication. *Journal of Virology*, 72(9):7510–7522, 1998.

[139] Yong Zhang, Yanbin Yin, Yunjia Chen, Ge Gao, Peng Yu, Jingchu Luo, and Ying Jiang. PCAS—a precomputed proteome annotation database resource. *BMC Genomics*, 4:42, 2003.

[140] M. Zuker. Computer prediction of RNA structure. *Methods in Enzymology*, 180:262–288, 1989.

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

VITA

Zasha Weinberg was born in Boston, USA, but spent most of his childhood in Toronto, Canada. In 1995, he earned a B.A. from New York University with a major in computer science, a minor in German and a suspicious number of Dramatic Writing courses in his first two years. After completing this degree, he achieved his life ambition: living by himself in an apartment in Manhattan's East Village. During this time, he worked as a computer programmer and manager. He moved to Seattle in 1999 to pursue a Ph.D. in the University of Washington's department of Computer Science & Engineering. This department awarded him an M.S. in 2001 and a Ph.D. in 2005.