Faster Genome Annotation of Non-coding RNA Families Without Loss of Accuracy

Zasha Weinberg[*] and Walter L. Ruzzo[†]

January 15, 2004

Abstract

Non-coding RNAs (ncRNAs) are functional RNA molecules that do not code for proteins. Covariance Models (CMs) are a useful statistical tool to find new members of an ncRNA gene family in a large genome database, using both sequence and, importantly, RNA secondary structure information. Unfortunately, CM searches are slow. This paper shows how to make CMs faster while provably sacrificing none of their accuracy. Specifically, based on the CM, our software builds a profile hidden Markov model (HMM), which filters the genome database. This HMM is a *rigorous filter*, i.e., its filtering eliminates only sequences that provably could not be annotated as homologs. The CM is run only on what remains. Optimizing the HMM for filtering involves minimizing an exponential objective function with linear inequality constraints. For most known ncRNA families, this allows an 8-gigabase database to be scanned in 2-20 days instead of years, and yields new family members missed by other techniques to improve CM speed.

Keywords: Non-coding RNA, gene families, covariance models, genome annotation, profile hidden Markov models, Iron Response Element, hyperthermophile archaea snoRNA, Histone Downstream Element, rigorous filter

[*]Dept. of Computer Science & Engineering, University of Washington, Box 352350, Seattle, WA, USA, 98195, zasha@cs.washington.edu

[†]Depts. of Computer Science & Engineering and Genome Sciences, University of Washington, Box 352350, Seattle, WA, USA, 98195, ruzzo@cs.washington.edu

1 Introduction

Non-coding RNAs (ncRNAs) are functional RNA molecules that do not code for proteins, e.g., tRNAs and spliceosomal RNAs. Recent work reveals that ncRNAs are much more numerous and significant than previously thought [27, 16, 5], e.g., with small nucleolar RNAs that process other ncRNAs [2], and many ncRNAs involved in regulating other genes such as microRNAs and analogous bacterial ncRNAs [17, 13, 28, 25]. Some mRNAs contain regulatory structural elements [15, 18] that can be viewed as ncRNAs.

To exploit prior work on the over 100 known ncRNA families, it is useful to annotate genomes with family homologs. Since secondary structure is often functionally important to RNAs, this task requires modeling both sequence and secondary structure. Techniques for finding ncRNA family members include searching for patterns that can include base pairing [24, 3, 15], which was used to find lentivirus Rev Response Elements [20], and searching for specific types of ncRNA, such as tRNAs [22, 11, 26], microRNAs [21] and small nucleolar RNAs [23, 9]. These methods require significant expert input, making them difficult to extend to new ncRNA families.

For the general problem of defining ncRNA families and finding new members, two methods exist that require modest manual work per family: Covariance models [8, 4] and ERPIN [12]. Both require only a multiple alignment of the family's members annotated with a secondary structure. From this input, a statistical model is built that is used to search a genome database. In tests, both techniques exhibit high sensitivity and selectivity on, e.g., tRNAs [12, 22]. A limitation of ERPIN is that it cannot accommodate non-consensus bulges in helices (which CMs can). Additionally, to prune its search, ERPIN sometimes requires the user to specify score thresholds for each helix, thus requiring more expert input and/or compromising accuracy. A limitation of CMs is that they cannot represent pseudoknots (which ERPIN can). It is not clear which limitation is more significant, but studies suggest that pseudoknots contain little information [8], whereas indels are common in many contexts. This raises a serious limitation of CMs: scans are very slow.

This paper seeks to address the impractical speed of CMs without sacrificing their accuracy. CMs are used in the Rfam Database [14] to annotate an 8-gigabase genome database called RFAMSEQ for over 100 ncRNA families. CMs are too slow to be used directly: e.g., searching RFAMSEQ to find tRNAs would take about 1 year on a 2.8 GHz Intel Pentium 4 PC. Obviously, this is improving as computers get faster, but there are over 100 families. Moreover, new families continue to be found and sequence data expands rapidly. To improve speed, Rfam uses a BLAST-based heuristic [1]. For each ncRNA family, the known members are BLASTed against RFAMSEQ, and the full CM is run only on the matches returned from BLAST. These searches are acceptably fast, but the BLAST heuristic—even with permissive settings—may cause family members to be missed that would be found with a regular (slower) CM search [14].

We develop a *rigorous filter*. Unlike a heuristic filter such as Rfam's use of BLAST, a rigorous filter guarantees that all sequences classified as homologs by the CM will be found; a rigorous filter will never increase the false negative rate over that of the CM. For our rigorous filter, a profile HMM is built from the CM, and run against the database. Based on the output, much of the database can be eliminated as provably not containing any family members that would be detected by the CM. The CM is run only on what remains.

Many HMM parameter settings guarantee rigorous filtering, but some settings eliminate more of the database than others. Under the simplifying assumption that genome sequences are adequately described by 0th-order Markov models, we optimize the parameters 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 our experiments, the profile HMMs search on average over 200 times faster than CMs. We tested 139 ncRNA families in Rfam version 5.0 (all families except those using the Rfam local alignment feature; see section 5). 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.

In pilot experiments, for some families, no new members were found over the BLAST heuristic. Since our filtering is rigorous, this means that if homologs were missed, finding them will require going beyond CMs as they are now. For other families, we find many homologs missed by the BLAST heuristic. E.g., a small nucleolar RNA of *Pyrococcus* is found to have putative homologs in a variety of other hyperthermophilic archaea.

The next section summarizes the results. Section 3 describes simplified CMs, with our technique to build and optimize HMMs in section 4. Then, we conclude and discuss future work. The appendix shows how our algorithm works on more realistic CMs and on standard CMs as in the literature.

Refer to `http://bio.cs.washington.edu/supplements/zasha-RECOMB-2004/` for supplementary information.

2 Results

2.1 Two kinds of profile HMM

Our 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 is discussed in appendix section 2.

2.2 Speed

Our 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 efficiency.* A fraction of 0 is perfect; 1 is pointless. We tested the filtering efficiency 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 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, 214 times for expanded type). The average overall speedup in our experiments was 0.04; i.e., if our technique were used for all of Rfam, it would be 25 times faster than using CMs. But, taking only the 127 families with filtering fractions $< 10^{-2}$, the speedup factor is 283.

2.3 Buried treasures

We scanned the full RFAMSEQ sequence for selected Rfam families (Table 2.3). For some families, no additional homologs were found compared to the Rfam database; given our technique's guarantees, it follows that the BLAST heuristic is sufficient for these fam-

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

Table 1: Filtering efficiency 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.

ilies. We biased our selection to families that we expected to challenge BLAST, e.g., short ncRNAs with low sequence identity, so we expect that BLAST will work for most other families, which are presumably easier. However, we also avoided families with poor filtering efficiency. Since neither BLAST nor HMMs use secondary structure, we expect BLAST to do worse for those families.

For many families, new hits (i.e., putative homologs) were found with the rigorous filter, as summarized in Table 2.3. Since our filter makes the CMs more sensitive, some of these may be false positives biologically, although these false positives may aid in tuning the CMs.

In fact, most of the new hits are biologically plausible, and we uncover new hits 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 two new hits in bacteria 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 [10]. The histone downstream element (RF00032), usually in metazoa, is found in various new metazoa and, intriguingly, in three plant species. 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.

Some hits missed by the BLAST heuristic are supported by an annotation, e.g., the U4 snRNA in green algae and a fungus, the histone downstream element in histone genes in five fly species, one marine invertebrate and two crustaceans, and the iron response element in ferritin genes in two aquatic invertebrates.

To see if BLAST could easily be adjusted to find all new hits, we 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

Rfam Id	**name**	**avg len**	**% id**	**# known**	**# new**	**HMM types used**	**frac. pred.**	**frac. actual**	**HMM +CM time (days)**	**est. CM time (days)**
RF00008 (Rfam 4.0)	Hammerhead ribozyme	35	68	313	278	C,E	1.1e-3	6.1e-4	1.4	97
RF00008	Hammerhead type 1	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	2	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 element	29	67	201	121	C,E	7.7e-4	7.7e-4	1.0	52
RF00043	Plasmid copy control	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 type 3	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	54	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

Table 2: Results of rigorous filtering experiments of RFAMSEQ

Each row in this table, except the first, is an Rfam 5.0 family. The first column is its Rfam accession Id. 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 our technique found. For HMM types used, C=compact type, E=expanded type. "C,E" means that an 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 $1.1 \cdot 10^{-3}$. 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 estimated time for a pure CM scan of RFAMSEQ.

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 gene families.

3 Simplified Covariance Models

Covariance Models (CMs) are statistical models that can detect when positional sequence and secondary structure resembles a given multiple RNA alignment. For simplicity, we discuss multiloops (CM bifurcation states) and nucleotides inserted into the consensus alignment in appendix section 1. We explain CMs somewhat unconventionally in terms of stochastic context-free grammars (SCFGs); appendix section 2 ties our explanation to the conventional view of CMs and corrects a subtle limitation of our pedagogical model. Readers unfamiliar with context-free grammars may find [4, chpt. 9] helpful.

3.1 Covariance Models are context-free grammars

Consider RNA molecules with sequence CAG or GAC with the C,G bases paired. A context-free grammar (CFG) for this is $\mathrm{S}_1 \rightarrow c\mathrm{S}_2 g | g\mathrm{S}_2 c$ and $\mathrm{S}_2 \rightarrow a$. (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 $c\mathrm{S}_2 g$ or $g\mathrm{S}_2 c$. Thus, we can produce the string CAG by the following steps, beginning with the start state: $\mathrm{S}_1 \rightarrow c\mathrm{S}_2 g \rightarrow cag$. The sequence of steps from the start state to an RNA sequence is called a *parse*.

CMs have states $\mathrm{S}_1, \mathrm{S}_2, \ldots, \mathrm{S}_n$ 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 [8, 6, 7].

All rules must be of the form $\mathrm{S}_i \rightarrow x_L \mathrm{S}_{i+1} x_R$, 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 $x_L = \varepsilon$ or $x_R = \varepsilon$ or both, the rule emits an unpaired nucleotide or no nucleotide; such rules can accommodate missing consensus positions and single-stranded regions.

Figure 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 $\Pr(\mathrm{S}_1 \rightarrow c\mathrm{S}_2 g) = 0.25$ and $\Pr(\mathrm{S}_2 \rightarrow a) = 1$, then the probability of the parse $\mathrm{S}_1 \rightarrow c\mathrm{S}_2 g \rightarrow cag$ 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 [8, 4]. 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.

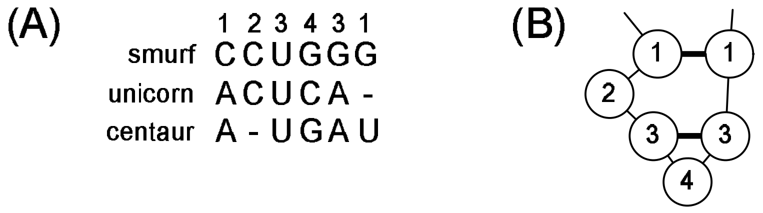

Figure 1: RNA multiple alignment, structure and CFG
(A) A multiple alignment of three hypothetical RNA sequences. Dashes (-) indicate missing nucleotides. (B) The structure. Thick lines are conserved base-pairs. Numbers refer to alignment positions; positions 1 and 3 are base paired, so appear twice. Position 1 is missing a base in unicorn.
A CM-style CFG that encodes these sequences and structures is $\mathrm{S}_1 \to a\mathrm{S}_2\varepsilon | a\mathrm{S}_2u | c\mathrm{S}_2g$; $\mathrm{S}_2 \to \varepsilon\mathrm{S}_3\varepsilon | c\mathrm{S}_3\varepsilon$; $\mathrm{S}_3 \to u\mathrm{S}_4a | u\mathrm{S}_4g$; $\mathrm{S}_4 \to c\mathrm{S}_5\varepsilon | g\mathrm{S}_5\varepsilon$; $\mathrm{S}_5 \to \varepsilon$. Note that normally CMs use less rigid grammars that allow anomalous nucleotides (with lower probability). A parse of the unicorn sequence is $\mathrm{S}_1 \to a\mathrm{S}_2\varepsilon \to ac\mathrm{S}_3\varepsilon\varepsilon \to acu\mathrm{S}_4a\varepsilon\varepsilon \to acuc\mathrm{S}_5\varepsilon a\varepsilon\varepsilon \to acuc\varepsilon\varepsilon a\varepsilon\varepsilon = acuca$.

4 Construction of the profile HMM from a simplified CM

Given a CM, we create a profile HMM 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. We describe HMMs in terms of stochastic regular grammars. First, we describe how we will use the profile HMM as a filter, then we explain the form of regular grammars allowed, then show how to covert 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, we show how to optimize scores to filter efficiently, subject to the constraints. The distinction between compact- and expanded-type HMMs is relevant only to the CMs in their full generality, not our pedagogical model; the distinction is described in the appendix.

4.1 Filtering with the profile HMM

Using the HMM, we compute 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. The window's length is a parameter that is part of the CM algorithm; a user must specify a window length for each family that is longer than any ncRNA family member could plausibly be. 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.

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

The key challenge is when $x_R \neq \varepsilon$ 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 AG. This sacrifice is a necessary consequence of using profile HMMs.

In general, a CM state S_i is expanded into a *left HMM state* $\overline{\mathrm{S}}_i^L$ and a *right HMM state* $\overline{\mathrm{S}}_i^R$. All CM rules $\mathrm{S}_i \rightarrow x_L \mathrm{S}_{i+1} x_R$, regardless of the value of x_L and x_R, are converted into HMM rules $\overline{\mathrm{S}}_i^L \rightarrow x_L \overline{\mathrm{S}}_{i+1}^L$ and $\overline{\mathrm{S}}_i^R \rightarrow x_R \overline{\mathrm{S}}_{i-1}^R$. (Note that the subscript on $\overline{\mathrm{S}}^R$ is decremented, since the right nucleotides are emitted in reverse order.) If $i = 1$, then we omit $\overline{\mathrm{S}}_{i-1}^R$. 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. See Table 3 for an example.

4.3 Constraints on HMM logarithmic scores

In the previous subsection, we showed how to create an HMM grammar corresponding to a CM grammar. As a first step to assigning logarithmic scores to HMM rules, we 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.

We 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. 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 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 $\mathrm{S}_i \rightarrow x_L \mathrm{S}_{i+1} x_R$ 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 1 has the rule $\mathrm{S}_3 \rightarrow u\mathrm{S}_4 a$ and corresponding HMM rules $\overline{\mathrm{S}}_3^L \rightarrow u\overline{\mathrm{S}}_4^L$ and $\overline{\mathrm{S}}_3^R \rightarrow a\overline{\mathrm{S}}_2^R$. Similarly, $\mathrm{S}_3 \rightarrow u\mathrm{S}_4 g$ corresponds to $\overline{\mathrm{S}}_3^L \rightarrow u\overline{\mathrm{S}}_4^L$ (again) and $\overline{\mathrm{S}}_3^R \rightarrow g\overline{\mathrm{S}}_2^R$. Let l_1 be the logarithmic score for $\overline{\mathrm{S}}_3^L \rightarrow u\overline{\mathrm{S}}_4^L$, l_2 for $\overline{\mathrm{S}}_3^R \rightarrow a\overline{\mathrm{S}}_2^R$ and l_3 for $\overline{\mathrm{S}}_3^R \rightarrow g\overline{\mathrm{S}}_2^R$, and suppose the score of CM rule $\mathrm{S}_3 \rightarrow u\mathrm{S}_4 a$ is -1 and $\mathrm{S}_3 \rightarrow u\mathrm{S}_4 g$ 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

CM state rules	**left HMM state rules**	**right HMM state rules**
$\mathrm{S}_1 \to a\mathrm{S}_2\varepsilon \vert a\mathrm{S}_2 u \vert c\mathrm{S}_2 g$	$\overline{\mathrm{S}}_1^L \to a\overline{\mathrm{S}}_2^L \vert c\overline{\mathrm{S}}_2^L$	$\overline{\mathrm{S}}_1^R \to \varepsilon \vert g \vert u$
$\mathrm{S}_2 \to \varepsilon\mathrm{S}_3\varepsilon \vert c\mathrm{S}_3\varepsilon$	$\overline{\mathrm{S}}_2^L \to \varepsilon\overline{\mathrm{S}}_3^L \vert c\overline{\mathrm{S}}_3^L$	$\overline{\mathrm{S}}_2^R \to \varepsilon\overline{\mathrm{S}}_1^R$
$\mathrm{S}_3 \to u\mathrm{S}_4 a \vert u\mathrm{S}_4 g$	$\overline{\mathrm{S}}_3^L \to u\overline{\mathrm{S}}_4^L$	$\overline{\mathrm{S}}_3^R \to a\overline{\mathrm{S}}_2^R \vert g\overline{\mathrm{S}}_2^R$
$\mathrm{S}_4 \to c\mathrm{S}_5\varepsilon \vert g\mathrm{S}_5\varepsilon$	$\overline{\mathrm{S}}_4^L \to c\overline{\mathrm{S}}_5^L \vert g\overline{\mathrm{S}}_5^L$	$\overline{\mathrm{S}}_4^R \to \varepsilon\overline{\mathrm{S}}_3^R$
$\mathrm{S}_5 \to \varepsilon$	$\overline{\mathrm{S}}_5^L \to \overline{\mathrm{S}}_4^R$	

Table 3: Example of converting an CM to a profile HMM
The CM grammar of Figure 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.

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 our 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, we 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.

4.4 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, we consider a heuristic to improve filtering. Regardless of the heuristic to improve filtering, we still require that the constraints are satisfied, so that rigorous filtering is guaranteed.

To improve filtering, we need a method to estimate an HMM's likely filtering efficiency. 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.

4.4.1 The infinite-length forward algorithm score

We develop a more practical heuristic to estimate filtering efficiency in a series of steps. First, we 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, we 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. We 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 our 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 our simplified CMs, which do not allow for inserted nucleotides, sequences beyond a certain length will have odds ratios of 0.)

Our 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. We take the opposite step, and use the forward algorithm to approximate the Viterbi in estimating an HMM's filtering efficiency.

Thus, our heuristic is to use the expected HMM forward algorithm result over database subsequences of unbounded length, generated from a 0th-order model. We 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.

We 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 sequence 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 sequence. Since we are calculating an expected sum for 0th-order sequences, our expected sum E is

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

where x^π_k is the kth nucleotide of x^π, $\Pr(x^\pi_k)$ 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 $P(\overline{\mathrm{S}}_{i\pm1} \to x_L\overline{\mathrm{S}}_i)$ 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). (Note that the forward algorithm does not use logarithms, so the logarithmic score variables must be exponentiated, e.g., $P(\overline{\mathrm{S}}_3^L \to u\overline{\mathrm{S}}_4^L) = 2^{l_1}$, using l_1 from section 4.3.) Then we obtain the recursion:

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

where $\mathrm{Pr}(x_L)$ is the 0th-order model's probability of nucleotide x_L, and $\mathrm{Pr}(\varepsilon) = 1$.

Gradients are used by many powerful and generic optimization algorithms. 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 from section 4.3).

Empirically, the forward algorithm heuristic 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.

4.4.2 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[19], 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.

4.5 Our algorithm on fully general CMs

The foregoing ideas can be extended to CMs in general, i.e., more sophisticated than our pedagogical model, by creating extra HMM states and rule types that correspond to the additional CM features. This allows inequalities to be created, and, with some tweaks, the infinite-length forward algorithm can handle the more complex HMMs. These technical details are left to the appendix.

5 Conclusions and future work

In terms of future work, one direction is to improve the filtering efficiency further. Work in progress extends the profile HMM to use limited secondary structure information to improve filtering, making rigorous filtering practical on more ncRNA families.

We also expect that CMs will be extended to improve their accuracy and versatility, and our approach could likely be extended to handle these improvements. One recent extension to CMs has been the *local alignment* feature [7], which allows a match to a part of the ncRNA, and is intended to detect homologs of ncRNA domains. We can model CM states with local alignments in terms of properties of left and right HMM states, with upper bound guarantees. However, without implementation, the effect on filtering efficiency is unknown.

In conclusion, covariance models are useful in annotating genomes with occurrences of known ncRNA gene families, but their slow speed has created practical problems. We have designed an algorithm that significantly speeds up scanning for a majority of ncRNA families in Rfam 5.0, with guarantees that no additional homologs will be missed. Our technique has revealed homologs missed by a previous technique, and gives a clearer picture of how well CMs perform in annotating genomes.

6 Acknowledgements

This material is based upon work supported in part by the National Institutes of Health under grants R01 HG02602 and NIH HG-00035 and by the National Science Foundation under grant NSF DBI-9974498. We thank Zizhen Yao and the anonymous referees for helpful comments on the manuscript.

References

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

[2] Decatur and Fournier. RNA-guided nucleotide modification of ribosomal and other RNAs. *Journal of Biological Chemistry*, 278(2):695–698, 2003.

[3] M. Dsouza, N. Larsen, and R. Overbeek. Searching for patterns in genomic data. *Trends in Genetics*, 13(12):497–498, 1997.

[4] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis*. Cambridge University Press, Cambridge, UK, 1998.

[5] S. R. Eddy. Computational genomics of noncoding RNA genes. *Cell*, 109:137–140, 2002.

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

[7] S. R. Eddy. *Infernal User's Guide*, 2003. `ftp://ftp.genetics.wustl.edu/pub/eddy/software/infernal/Userguide.pdf`.

[8] S. R. Eddy and R. Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Research*, 22(11):2079–2088, 1994.

[9] S. Edvardsson, P. P. Gardner, A. M. Poole, M. D. Hendy, D. Penny, and V. Moulton. A search for H/ACA snoRNAs in yeast using MFE secondary structure prediction. *Bioinformatics*, 19(7):865–873, 2003.

[10] G. Ferbeyre, V. Bourdeau, M. Pageau, P. Miramontes, and R. Cedergren. Distribution of hammerhead and hammerhead-like RNA motifs through the GenBank. *Genome Research*, 10(7):1011–1019, 2000.

[11] G. Fichant and C. Burks. Identifying potential tRNA genes in genomic DNA sequences. *Journal of Molecular Biology*, 220(3):659–671, 1991.

[12] D. Gautheret and A. Lambert. Direct RNA motif definition and identification from multiple sequence alignments using secondary structure profiles. *Journal of Molecular Biology*, 313:1003–1011, 2001.

[13] S. Gottesman. Stealth regulation: biological circuits with small RNA switches. *Genes and Development*, 16:2829–2842, 2002.

[14] S. Griffiths-Jones, A. Bateman, M. Marshall, A. Khanna, and S. R. Eddy. Rfam: an RNA family database. *Nucleic Acids Research*, 31(1):439–441, 2003. `http://rfam.wustl.edu`.

[15] G. Grillo, F. Licciulli, S. Liuni, E. Sbisà, and G. Pesole. PatSearch: a program for the detection of patterns and structural motifs in nucleotide sequences. *Nucleic Acids Research*, 31(13):3608–3612, 2003.

[16] A. Hüttenhofer, J. Brosius, and J.-P. Bachellerie. RNomics: identification and function of small, non-messenger RNAs. *Current Opinion in Chemical Biology*, 6:835843, 2002.

[17] D. Kennedy. Breakthrough of the year. *Science*, 298(5602):2283, 2002.

[18] E. C. Lai. RNA sensors and riboswitches: Self-regulating messages. *Current Biology*, 13:R285–R291, 2003.

[19] C. Lawrence, J. L. Zhou, and A. 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.

[20] E. A. Lesnik, R. Sampath, and D. J. Ecker. Rev response elements (RRE) in lentiviruses: an RNAMotif algorithm-based strategy for RRE prediction. *Medicinal Research Reviews*, 22(6):617–636, 2002.

[21] L. P. Lim, N. C. Lau, E. G. Weinstein, A. Abdelhakim, S. Yekta, M. W. Rhoades, C. B. Burge, and D. P. Bartel. The microRNAs of *Caenorhabditis elegans*. *Genes and Development*, 17(8):991–1008, 2003.

[22] T. Lowe and S. R. Eddy. tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence. *Nucleic Acids Research*, 25(5):955–64, 1997.

[23] T. M. Lowe and S. R. Eddy. A computational screen for methylation guide snoRNAs in yeast. *Science*, 283:1168–1171, 1999.

[24] T. J. Macke, D. J. Ecker, R. R. Gutell, D. Gautheret, D. A. Case, and R. Sampath. RNAMotif, an RNA secondary structure definition and search algorithm. *Nucleic Acids Research*, 29(22):4724–4735, 2001.

[25] E. G. Moss. MicroRNAs: hidden in the genome. *Current Biology*, 12(4):R138–140, 2002.

[26] 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.

[27] G. Storz. An expanding universe of noncoding RNAs. *Science*, 296(5571):1260–1263, 2002.

[28] E. Wagner and K. Flardh. Antisense RNAs everywhere? *TRENDS in Genetics*, 18(5):223–226, 2002.

APPENDIX

1 Covariance models with bifurcation and insertion states

RNA multiloops are handled by CM bifurcation states, which permit rules like $\mathrm{S}_i \rightarrow \mathrm{S}_j\mathrm{S}_k$ 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.

Our 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 $\mathrm{IL}_i \rightarrow x_L\mathrm{IL}_i$, or on the right with $\mathrm{IR}_i \rightarrow \mathrm{IR}_i x_R$ ($x_L, x_R \neq \varepsilon$). Any state S_i may transition to an IL_i state with rule $\mathrm{S}_i \rightarrow x_L\mathrm{IL}_{i+1}x_R$. 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 $\overline{\mathrm{IL}}_i^L$ (no right HMM state is needed). Similarly IR_i maps to $\overline{\mathrm{IR}}_i^R$. A rule $\mathrm{S}_i \rightarrow x_L\mathrm{IL}_i x_R$ becomes three HMM rules. The left rule is $\overline{\mathrm{S}}_i^L \rightarrow x_L\overline{\mathrm{IL}}_i^L$. 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 we add two HMM rules: $\overline{\mathrm{S}}_i^R \rightarrow x_R\overline{\mathrm{IR}}_i^R$ and $\overline{\mathrm{S}}_i^R \rightarrow x_R\overline{\mathrm{S}}_{i-1}^R$ (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 $\overline{\mathrm{IL}}_i^L \rightarrow x_L\overline{\mathrm{IL}}_i^L$. The exit rule $\mathrm{IL}_i \rightarrow \mathrm{S}_{i+1}$ becomes $\overline{\mathrm{IL}}_i^L \rightarrow \overline{\mathrm{S}}_{i+1}^L$. 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 $\overline{\mathrm{IL}}_i^L \rightarrow x_L\overline{\mathrm{IL}}_i^L$ 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 $\overline{\mathrm{IL}}_i^L \rightarrow x_L\overline{\mathrm{IL}}_i^L$, 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.

2 Our technique in conventional CM terms

Readers who are familiar with CMs, e.g., from [4, chpt. 10], will notice that we have explained them in an unusual way. Moreover, our pedagogical version of CMs is not quite as powerful as true CMs. In this section, we show how our algorithm fully supports CMs

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

Table 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.

as they are presented in the literature. In fact, the computer program we wrote to perform our experiments uses the source code of the Infernal package (`infernal.wustl.edu`), which implements CM searches for the Rfam database.

The form of SCFGs we 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, our 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* [6] 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 — $\overline{\mathrm{S}}_i^L$, $\overline{\mathrm{S}}_i^R$, $\overline{\mathrm{IL}}_i^L$ and $\overline{\mathrm{IR}}_i^R$ in our earlier explanation. The $\overline{\mathrm{S}}_i^L$ state must be split into two states, corresponding to ML and D types, and the same for the $\overline{\mathrm{S}}_i^R$ 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 $\overline{\mathrm{S}}_i^L$ or $\overline{\mathrm{S}}_i^R$ state.

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 summarizes the HMM states in a CM MATP node.

In making constraints for the HMM scores, we 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 sub-paths, and create one linear inequality for each sub-path.

Finally, our 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.