7-1-2009

RNA Search with Decision Trees and Partial Covariance Models

Jennifer A. Smith
Boise State University

DOI: 10.1109/TCBB.2008.120

RNA Search with Decision Trees and Partial Covariance Models

Jennifer A. Smith

Abstract—The use of partial covariance models to search for RNA family members in genomic sequence databases is explored. The partial models are formed from contiguous subranges of the overall RNA family multiple alignment columns. A binary decision-tree framework is presented for choosing the order to apply the partial models and the score thresholds on which to make the decisions. The decision trees are chosen to minimize computation time subject to the constraint that all of the training sequences are passed to the full covariance model for final evaluation. Computational intelligence methods are suggested to select the decision tree since the tree can be quite complex and there is no obvious method to build the tree in these cases. Experimental results from seven RNA families shows execution times of 0.066-0.268 relative to using the full covariance model alone. Tests on the full sets of known sequences for each family show that at least 95 percent of these sequences are found for two families and 100 percent for five others. Since the full covariance model is run on all sequences accepted by the partial model decision tree, the false alarm rate is at least as low as that of the full model alone.

Index Terms—Bioinformatics, computational intelligence, covariance models, decision trees, RNA database search.

1 Introduction

Searching for new members of RNA sequence families in genomic data is made more difficult than protein sequence homology search by the need to account for secondary structure. Searches based on primary sequence only are generally not very effective since base-pairing patterns (secondary structure) are much more highly conserved than nucleotide identity in functional RNA [1], [2]. The underlying reason is that RNA molecules which perform such functions as catalysis or molecular recognition require a specific three-dimensional shape in order to perform these tasks. The most important factor in determining three-dimensional shape is the base-pairing pattern. This pattern is often maintained by substituting pairs of nucleotides simultaneously such that the pair remains canonical (a GC, AU, or GU pair in either order), although exceptions to the need for canonical pairs do exist.

A modeling strategy that includes joint probabilities of substitution for both nucleotides of a consensus pair is the covariance model [3]. Like its cousin, the profile hidden Markov model [4] used for primary sequence modeling, the covariance model includes position-specific penalties for insertions, deletions, and substitutions. Covariance model estimation and search has been implemented by the Infernal [5] package which is the basis for creating the Rfam [6] RNA families database. A major problem with Infernal is that it is extremely computationally demanding. The search algorithm uses dynamic programming for optimal alignment and scoring, but often requires sequence prefiltering in order to get acceptable computation times. Early use of the search algorithm relied on a BLAST [7] primary sequence prefilter to find database regions to search with the full covariance model. More recently, hidden Markov model (HMM) primary sequence prefilters have been employed either as rigorous filters [8] or nonrigorous filters [9]. The former guarantee that the prefilter will not remove any database region which would exceed the covariance model threshold. The rigorous filters often do not reduce the database size sufficiently, and nonrigorous HMM filters, like the BLAST filters, can erroneously reject database regions. There have also been recent speed improvements obtained by excluding very improbable regions of the alignment search space, such as the application of query-dependent banding (QDB) [10]. These methods usually involve limiting the net number of insertions and deletions allowed on subsequences of the model's consensus sequence. Other methods of searching for noncoding RNA genes include methods specific to a particular class of ncRNA such as RNAmmer [23] which is specifically targeted at ribosomal RNA and methods which use only a single sequence rather than a family multiple alignment such as RSEARCH [25]. A combined package that can use either the rigorous or nonrigorous filters discussed above is available as RaveNnA [24].

Crucial to keeping computation times feasible is limiting the number of net insertions allowed in new family members which are sought in the database. The model has a consensus sequence length (length of the sequence with highest possible score) which is added to whenever a database sequence has insertions with respect to the consensus sequence and subtracted from whenever the database sequence has deletions. The original search method of Infernal used a maximum search length D specified in advance. This length maximum was applied to all subsequences throughout the scoring procedure. This resulted in some significant inefficiencies when scoring

• *The author is with the Electrical and Computer Engineering Department, Boise State University, 1910 University Ave., Boise, ID 83725-2075. E-mail: jasmith@boisestate.edu.*

Manuscript received 11 May 2008; revised 21 Oct. 2008; accepted 22 Oct. 2008; published online 29 Oct. 2008.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBB-2008-05-0086.
Digital Object Identifier no. 10.1109/TCBB.2008.120.

very short consensus subsequences near the model ends. These inefficiencies have been largely eliminated in the recent QDB versions of Infernal since there is now an upper and a lower bound on search length that varies with the consensus subsequence being scored. The computational complexity of the original search method was $O(LDM_{nb} + LD^2M_b)$, where L is the length of the database to search, D is the maximum search length, M_{nb} is the number of nonbifurcation states, and M_b is the number of bifurcation states. A discussion of covariance model states appears in a later section, but it is clear that reducing D reduces computation time given a particular model (M values) and database (L value) at the risk of possibly missing long RNA family members in the database. It is also clear that elimination of bifurcation states is more effective at reducing computation time than elimination of the same number of nonbifurcation states. The parameter D needs to be at least as large as the consensus sequence length and the number of model states increases approximately in proportion to consensus sequence length. As a result, the necessary computation time tended to increase faster than proportionally to the square of the consensus sequence length in the original method.

With QDB, it is the range of reasonable search lengths for each consensus subsequence that is important. The model parameters themselves tell when the accumulated insertion and deletion penalties needed to get a particular deviation from the consensus subsequence length make reaching the score threshold impossible (or at least highly unlikely). As a result, it is not as clear how computation time varies with consensus sequence length. Longer subsequences generally have more opportunities for insertions and deletions and therefore typically have a larger range of search lengths. Whether these ranges of lengths usually increase more or less than proportionally with overall consensus sequence length is not well known. For the five RNA families studied in this paper, the answer appears to be less than proportionally. With or without QDB, it is expected that the sum of the computation times of several partial models will often be less than the computation time of a full model when the sum of the consensus lengths of the partial models equals that of the full model. This computation time reduction will be even more pronounced if breaking the full model into partial models removes the need for any bifurcation states (such a partition is always possible and is always used in the examples of this paper).

An even further reduction in computation time can be achieved if one avoids running all of the partial models on every portion of the database. The scores observed from partial models run earlier on a section of database can be used to chose which partial model to run next, including the possibility of completely skipping one or more partial models. In most cases, not running one or more partial models results from not attaining a high enough score on earlier partial models to make obtaining the overall score threshold more than very remotely possible (or perhaps not possible at all). Occasionally, not running further partial models may be the result of already exceeding the overall threshold before all partial models are run. It is proposed in this work that the choice of partial model order, partial model score interaction, and when to stop applying partial models and either reject or accept a database section could be described with a binary decision tree. Finding a good decision tree can be a complex task and a computational intelligence method for doing this is proposed.

The reduced computational time required by partial models relative to a full model does not come without a cost. In general, the product of the false alarm rates of the partial models is greater than that of the full model. This is the result of not penalizing insertions between the partial models and/or allowing partial models to overlap (effectively not forcing one or both of the partial models to take deletions in order to not share database positions). To make sure that the false alarm rate for the partial models is no more than for the full model, the full model will always be run on those portions of the database that are accepted by the partial model decision tree. As such, the partial models work as a type of prefilter. If the fraction of the database that is accepted by the decision tree is small, then the cost of running the full model will also be small. It must also be the case that true RNA family members are accepted by the decision tree, otherwise they will not be found. The objective of the decision tree design is therefore to minimize computation time subject to the constraint that all training sequences (known as "seed" sequences in the Rfam database) are accepted. The Rfam database also includes more sequences that have been found using the existing models and the hope is that a very high percentage of these test sequences also are accepted by the decision tree. In the experimental section of this paper, seven RNA families are examined. They are broken into somewhat equal sized partial models and decision trees found. The resulting computation time (including full model computation of all portions of the database accepted by the decision tree) is 0.066-0.268 times that of applying the full model to the whole database. By construction, the decision tree accepts all training sequences. Tests on the full sets of known sequences for the seven families result in 96 percent of U4 RNA family sequences being accepted, 95 percent of the RyhB sequences, and 100 percent for five other RNA families. Assuming the same full model threshold is used whether applied to the full database or only to the decision tree output, all sequences that are accepted by the decision tree will pass the full model test. The false alarm rate will be at least as low with the decision tree as without. False alarms in the accepted database portion will occur in exactly the same places, but false alarms cannot occur in the portions rejected by the decision tree. It is not valid to multiply the proportion of database accepted by the false alarm rate of the full model to obtain an overall false alarm rate. This is because database sections which cause a false alarm in the full model are much more likely to get through the decision tree than random sections of the database. So, no more can be said than that the false alarm rate is at least as low when using the partial model prefilter as when not.

The paper first gives a very quick overview of covariance models and how they may be used to search for RNA in genomic data (Section 2). The partitioning of the columns of a secondary-structure-annotated multiple alignment, the estimation of partial models, and the execution times of the

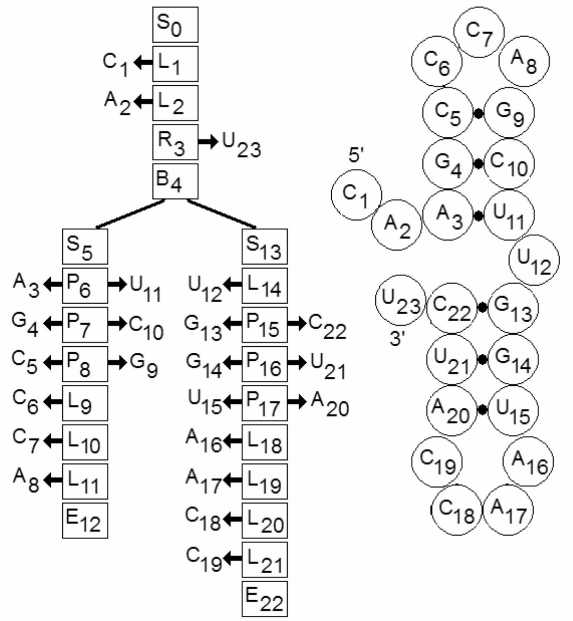

Fig. 1. Example covariance model node tree and associated RNA secondary structure diagram. Both show consensus structure (no insertions or deletions) and consensus sequence (most likely nucleotide at each consensus position).

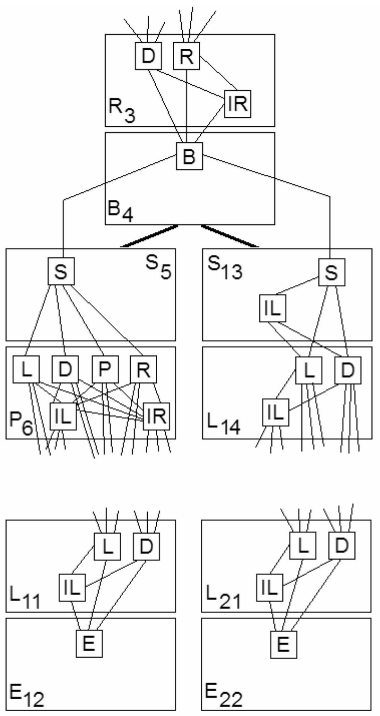

Fig. 2. Example covariance model states and interconnection for portions of the covariance model node tree. Large boxes are CM nodes from Fig. 1 and small boxes are the states internal to the nodes. Insert states (IL and IR) have identical function to consensus L and R states with the exception that the insert states have self-transition connections (not shown). S and D states also have identical function even though the reasons for their use are distinct.

partial models versus full models is examined for five RNA families (Section 3). The use and choice of binary decision trees to guide the application order and thresholds used with partial models is then presented (Section 4). Decision trees are found for the five RNA families and performance investigated (Section 5). Finally, conclusions are drawn (Section 6).

2 Covariance Models for RNA Search

2.1 Covariance Model Structure

A covariance model can be thought of as an extension of a profile hidden Markov model to incorporate expression of joint probabilities of base pair substitutions rather than just the marginal probabilities of each half of the pair individually. There are two classes of symbol-emitting nodes: one which specifies an unpaired consensus position and emits a symbol (A, C, G, or U) with a probability assigned to each and one which specifies a pair of consensus positions and emits a symbol pair (AA, AC, AG, ..., UU) with a probability assigned to each of the 16 possible pairs. The two halves of the consensus pair are generally not next to each other in the consensus sequence. Consensus refers to the nominal arrangement of nucleotides in sequences of a particular RNA family from which individual sequences may deviate through insertions, deletions, or substitutions. A profile HMM can be viewed as simply a covariance model in which no base-pairing occurs.

Fig. 1 shows a secondary structure diagram of a simple example RNA on the right and the covariance model node tree associated with this consensus secondary structure on the left. The RNA sequence is represented starting at the $5'$ end (left end) and progressing to the $3'$ end (right end). In the diagram, the most likely (consensus) nucleotide at the left position (position index 1) is a C and is labeled C_1. If the ordered sequence of length 23 with symbols CAAGCCCAG... CCAUCU appears in the database at contiguous positions, it should generate the maximum possible score during database search for this RNA family. Twelve of the consensus positions (3-5, 9-11, 13-15, and 20-22) are base-paired with other positions.

The covariance model node tree models unpaired positions as either L or R nodes and paired positions as P nodes. The remaining node types (S, B, and E) are nonemitting and serve to organize the location of the three emitting node types (L, R, and P). An L node emits a symbol to the left of a subsequence represented by its child node. An R node emits a symbol to the right of the child subsequence. The P node emits a symbol on the right and the left simultaneously. It is the existence of P nodes which are the fundamental difference between a profile HMM and a covariance model. All of the consensus nucleotides from the secondary structure diagram also appear next to the covariance model node which models the position. In both cases, the nucleotide is merely the most likely one for the particular position and all possible nucleotides are assigned some nonzero probability at each position.

Internal to each node in the covariance model tree is a state structure which allows for position-specific insertion and deletion penalties as shown in Fig. 2. Penalties are also different for insertion or deletion continuations versus initiation. These penalties take the form of state transition probabilities. All probabilities (both emission and transition) are given in the form of log likelihood ratios such that

score calculations involve addition and comparison, but never multiplication.

2.2 Database Search Using Covariance Models

Once a covariance model has been estimated using a set of known sequences in an RNA family annotated with base-pairing information, the model may be used for database search. Given a position in the database at which the 3′ end of the sequence must be located and a maximum number of insertions net of deletions in the database sequence, the score of the database end position may be calculated relative to the covariance model. The standard solution to this scoring problem is to use dynamic programming. At each state in the model, the highest scoring combination of child state score, state transition score, and emission score (if the state is an emitting state) is chosen. The choice among state children is equivalent to choosing among subsolutions with insertions, deletions, or neither on the ends of the subsolutions.

The usual dynamic programming scoring algorithm is attributed to the works of Cocke, Younger, and Kasami, and is thus known as the CYK algorithm. End (E) states represent null sequences and are given a score of 0 when they are associated with a subsequence of length $d = 0$, but a score of $-\infty$ otherwise. Working from the E states toward the root start state, subsequences grow by addition of a symbol on the left end, right end, or both ends simultaneously with each visit to a left, right, or pair (L, R, or P) state, respectively. Delete (D) states allow bypass of the symbol-emitting L, R, or P states. The single-emission L and R states can be used to emit the consensus symbol (in an L or R node, respectively), to emit only half of a consensus pair (in a P node) or insertions relative to the consensus (where they are normally labeled IL or IR to highlight this usage). The IL and IR states are the only ones that include self-transitions (not shown in the figure). Start (S) states allow for the collection of several child solutions without emitting anything and are functionally identical to D states. Bifurcation (B) states join two contiguous child solutions into a whole.

Any given state in conjunction with all states below it can be thought of as a representation of a consensus subsequence with possible insertions and deletions. The score of the given state can be found for various lengths of this subsequence with insertions and deletions included. Before the use of QDB, all states were scored for all possible lengths in the range 0 to D. This results in scores calculated for some very improbable lengths (for example, scoring the L state in the L_{11} node for lengths very close to D can only be accomplished with many visits to IL first and will acquire a huge insertion penalty). QDB causes every state to have its own range of scoring lengths centered on the consensus subsequence length such that only lengths that incur reasonable aggregate insertion and deletion penalties are considered.

The entire search (without QDB) can be expressed as three nested loops over database end position j, search subsequence length d, and state number v (listed from outer to inner loop). The score of interest at each database position j is the maximum value over search length d at the root start state. That is, one wants $\max_d$ $[s(j, d, 1)]$ at each position j, where $s(j, d, v)$ is calculated using

$$
\begin{aligned}
&\text{for} \quad j = 0 : L, d = 0 : D(\text{s.t.} d \leq j),\ v = M : 1 \ \{ \\
&\quad y = \text{set of children of state } v \\
&\quad \mathrm{s}(j, d, v) = \\
&\quad \text{case state type of } v \text{ is } \{ \\
&\qquad \mathrm{E} : s(j, d = 0, v) = 0;\ s(j, d > 0, v) = -\infty \\
&\qquad \mathrm{D}, \mathrm{S} : \max_y [s(j, d, y) + t(v, y)] \\
&\qquad \mathrm{L} : \max_y [s(j, d-1, y) + t(v, y)] + e(j - d + 1) \\
&\qquad \mathrm{R} : \max_y [s(j-1, d-1, y) + t(v, y)] + e(j) \\
&\qquad \mathrm{P} : \max_y [s(j-1, d-2, y) + t(v, y)] + e(j, j - d + 1) \\
&\qquad \mathrm{B} : \max_{0 \leq k \leq d} [s(j-k, d-k, y_{\mathrm{L}}) + s(j, k, y_{\mathrm{R}})] \\
&\quad \} \\
&\}
\end{aligned}
$$

In cases where an index is negative, the associated score is made to be $-\infty$. State transition scores are given as $t(v, y)$ from child state y to parent state v. Emission scores for single emissions are given by $e(x)$, where x is a database sequence position that returns either an A, C, G, or U, and emission scores for paired positions is given by $e(x_L, x_R)$, where x_L and x_R are a pair of database sequence positions returning an ordered pair of database symbols. It can be seen that B states are particularly expensive to calculate since these scores require an addition loop over k with upper limit d, whereas all other states have no fourth inner loop. The resulting complexity is $O(LDM_{nb} + LD^2M_b)$, where M_{nb} is the number of nonbifurcation states and M_b is the number of bifurcation states. The QDB modification limits the range of d at each state. The QDB complexity then becomes $O(LD_{nb}M_{nb} + LD_b^2M_b)$, where D_{nb} is the mean nonbifurcation state search bandwidth and D_b^2 is the mean of the squared search bandwidths for bifurcation states.

3 Partitioning and Partial Models

When building a covariance model from aligned sequence data, it is necessary to have a contiguous range of sequence positions. However, one cannot just choose any range of alignment columns to build a partial model since any base-paired position in the range must have the other half of the base pair also in the range. Dividing the alignment between the two children of a covariance model bifurcation will always work. It is also always possible to divide the columns at any point that is not enclosed by any base pair. In the example model of Fig. 1, the division between U_{11} and U_{12} is clearly possible since U_{11} is the rightmost consensus position of the bifurcation's left child and U_{12} is the leftmost consensus position of the bifurcation's right child. Alternatively, one notes that C_1, A_2, U_{12}, and U_{23} are not enclosed by any base pair, and therefore, a division can be made on either side of these consensus positions. Possible partial models include C_1 by itself, $\mathrm{C}_1\mathrm{A}_2$, and $\mathrm{G}_{13}\mathrm{G}_{14}\mathrm{U}_{15}\mathrm{A}_{16}\mathrm{A}_{17}\mathrm{C}_{18}\mathrm{C}_{19}\mathrm{A}_{20}\mathrm{U}_{21}\mathrm{C}_{22}$. Although the partial model $\mathrm{G}_{14}\mathrm{U}_{15}\mathrm{A}_{16}\mathrm{A}_{17}\mathrm{C}_{18}\mathrm{C}_{19}\mathrm{A}_{20}\mathrm{U}_{21}$ would be valid, it is not a good choice, since G_{13} and U_{22} cannot then be assigned to any other partial model partition. One also notes that U_{12} could have been assigned to either of the children of the bifurcation (as an R node at the top of the left child or as an L node at the top of the right child). The standard used by

```
ACCCU.CGUAUUAUUG.UGUUUUGGG.UUGAAACCGAGCUUAGCCUUGUU
ACCCC.GGUAUUAUUGCUGUUAUGGG.UUUGAUUCGAGCUUAGCCUUGUU
ACCCCCGCUGUUGUUGCUCUACUGGG.UUGCGAUCGAGCUUAGCCUCGUU
ACCCG.UUGUUUGUAGAUGUUAUGGGGUUUGG.CCGAGCUUAGCGUUGUU
ACACC.CGGUUUAUUGCUGUUGUAGG.GUUGG.CACAGCUUAGCCUUGUU
.......................<<<.<<<.....<<<....<<<<.<<<
11111111111111111111111122222222222222222222222222

CAUUUUUUUACACAACACCUCCGUAAAAA.AUAAAACACGGCGCACCGGA
CAUUUUUUUACACAACACCCCCGUAAAAA.AUACAACACGGCCCGC....
CAUUUUUUUAUACAACAGCCCGUUAAAAACAGUAGACA.GGC.CGC....
CAUUUUUUUACACAAUAGGCCUGUAAAAA.AUUGAACA.CGCACUC..CU
CAAUUUUUCGUACAACGGAGCCUGUAAAAUA.UCUGCU.GCACCAG..GU
<<.<<<<<<<<<.........>>>>>>>>.>.>>>>>>.>>>.>>>....
22222222222222222222222222222222222222222222222222

GAUAGCCCUUUACCCGGGAAAU....AGCGGAUAAUGGAUUGCCCCGAUA
UCAAG.CCUGCGGUACCCAAGGCGGCCGAGCCCCAAAAAACGCCCUGUUU
UUUAG.CUCUUAUUUAAGAUCGGUAAAAAACGGAAUAAAACGCCCUAAAU
GAACACCCUUCGACGCGAGGUUUU..UUUCGGCUUUUCGUCCGGCGAAAA
UGAACACCUUGCCUGGCCUGGCUU.UGGCCGCUUUCGCGAUGCGCUGGAA
..>>>.>>>.....................................>>>.
22222222233333333333333333333333333333333333333333

UGCAGCAAAUCUGCCGUAGAUGUGCCAUUUCAGGGCAU
UGCCCCAAAAGAGCCUUGCCAGCCCUUUAUUAGGGCAU
AGCAGCAAAAAAGGCCAUAUGGCGCAGUUUUAGGGCAA
UGCCCUGAAAACGGCCUCAACGCCCCACAAGAGGGCGU
GGCCCUGUCUACGGCCAUUUCGCCGGCCGCGAGGGCGA
..................<<<<<<<....>.>>>>>>
33333333333333333334444444444444444444
```

Fig. 3. Five sequences forming the "seed" alignment for the RF00623 (P1) RNA family in Rfam. The secondary structure annotation lines have a dot for unpaired sequence locations, < for the left half of a pair, and a > for the right half of a pair. Since the standard covariance model cannot handle pseudoknots (two or more base pairs that are neither nested nor disjoint), this notation is unambiguous. The seventh line of the alignment shows a possible assignment of columns to one of four partial models.

Infernal is to always use an L node whenever either an R or an L node is possible, and hence the choice made in Fig. 1. However, the choice of alignment columns for partial models does not need to follow this rule.

The two children of a bifurcation are always start nodes and merely renumbering nodes and states such that each child becomes a root start node almost gives two valid partial models. The only difference is that a root start node includes both left and right insert states whereas a left child start node includes neither type of insert state and a right child start node includes only a left insert state (see Fig. 2). This is done since the nodes above the bifurcation are responsible for the insertion possibilities outside of the joined bifurcation children. Insertion possibilities between the two joined children could be done with either an insert right on the left child or an insert left on the right child. Keeping with Infernal's preference for L to R rule, an IL state is assigned to the right child. The easiest way to generate partial model parameter files is to divide the multiple alignment file into contiguous ranges of columns and run the Infernal `cmbuild` program on the new input files rather than try to renumber the nodes and states and add new IR and possibly IL states to the start nodes.

An example of multiple alignment of the five seed sequences for the Rfam P1 RNA family (accession number RF00623) is shown in Fig. 3 along with a possible assignment of contiguous columns to one of four partial models (the numbers 1-4 in every seventh row). Partial model 1 is a 5′ dangling end of all L nodes. This partial model is conceptually identical to a profile HMM. Partial model 2 is a stem-loop (hairpin) structure as is partial model 4. Partial model 3 is a connector between the two hairpins and is also composed of all L nodes. To create the four partial model parameter files, simply create four multiple alignment files with the assigned alignment columns in each and run `cmbuild` four times.

Not all RNA families are amenable to division into parts. Some families have consensus sequences that are too short (perhaps only 20-30 consensus positions) to warrant division. Searches for these family members already run fast relative to the families with longer consensus sequences. Some other families have two halves of a base pair with one position very close to the 5′ end and the other very close to the 3′ end of the consensus sequence. These families can often be broken up into reasonable partial models, but the partial models do not fully cover the original full model. If the part of the full model that is not covered is highly conserved, then the partial models may not be very effective. There still remains a large portion of the longer family models that can be divided completely and the number of such families should continue to grow as new families are found.

Seven RNA families from Rfam have been divided into parts, as shown in Table 1: U4 (five parts), MicC (four parts), DsrA (three parts), HgcF (three parts), P1 (four parts), RyhB (three parts), and SL1 (three parts). The U4 family has a large number of seed sequences (30) and total number of known sequences (102 after filtering to less than 90 percent sequence identity). Because of the large number of member sequences, U4 will be chosen for detailed analysis in the following sections. For each partial model in Table 1, the consensus column numbers are shown. This is followed by two sets of execution time data normalized to the full model execution time. The data were generated using the UNIX `time` command (user time) when running the cmsearch program in the Infernal 0.81 package on a recent Pentium single processor. By default, Infernal 0.81 uses QDB, but this can be turned off with the $-noqdb$ flag. It can be seen that the advantage of breaking the full covariance model into pieces is less using QDB than without. Since using QDB generally makes search time about half as much as without, it is expected that most users will want to use QDB and all the results that follow in this paper are based on the assumption that QDB will be used. The "predicted" lines in Table 1 show the normalized values of D times M, where D is the automatically calculated maximum search length and M is the number of covariance model states. It can be seen that DM is a fairly good predictor of execution time when not using QDB. As expected, it is not so good at predicting execution times with QDB. Finally, the sums of the measured and predicted normalized execution times across all the partial models is shown. Particularly when using QDB, there is not a lot of advantage to the partial models if all partial models need to be run on every portion of the database. This is even more true if one is going to run the full model on any significant portion of the database that has been accepted by the partial model search in order to get more reasonable false alarm rates. Occasionally, running all partial models on the full database takes even longer than

TABLE 1
Partitions and Execution Times

Family	Full	Part 1	Part 2	Part 3	Part 4	Part 5	Sum
U4	1-137	1-16	17-56	57-79	80-113	114-137	
w/ QDB	1.000	0.042	0.181	0.091	0.213	0.120	0.647
w/o QDB	1.000	0.017	0.084	0.039	0.094	0.051	0.285
Predicted	1.000	0.023	0.102	0.045	0.103	0.066	0.340
MicC	1-121	1-21	22-37	38-70	71-121		
w/ QDB	1.000	0.139	0.074	0.268	0.443		0.924
w/o QDB	1.000	0.056	0.029	0.110	0.221		0.415
Predicted	1.000	0.045	0.030	0.098	0.199		0.373
DsrA	1-87	1-25	26-59	60-87			
w/ QDB	1.000	0.268	0.430	0.330			1.028
w/o QDB	1.000	0.123	0.204	0.154			0.481
Predicted	1.000	0.110	0.178	0.135			0.424
HgcF	1-168	1-30	31-96	97-168			
w/ QDB	1.000	0.141	0.402	0.403			0.946
w/o QDB	1.000	0.052	0.179	0.193			0.424
Predicted	1.000	0.045	0.173	0.202			0.420
P1	1-181	1-22	23-113	114-172	173-181		
w/ QDB	1.000	0.058	0.513	0.231	0.082		0.884
w/o QDB	1.000	0.022	0.254	0.109	0.028		0.413
Predicted	1.000	0.023	0.218	0.122	0.018		0.382
RyhB	1-84	1-23	24-53	54-84			
w/ QDB	1.000	0.202	0.377	0.239			0.818
w/o QDB	1.000	0.110	0.186	0.118			0.414
Predicted	1.000	0.112	0.178	0.117			0.407
SL1	1-103	1-33	34-68	69-103			
w/ QDB	1.000	0.321	0.318	0.311			0.950
w/o QDB	1.000	0.154	0.142	0.144			0.440
Predicted	1.000	0.151	0.143	0.147			0.441

Partitions are consensus column numbers in the seed (training) multiple alignment. Execution times are relative to that of the full model. Rfam accession numbers: U4 = RF00015, MicC = RF00121, DsrA = RF00014, HgcF = RF00058, P1 = RF00623, RyhB = RF00057, SL1 = RF00198. Predicted values are D times M, where D is the maximum search length and M is the number of states. Execution times are shown both with (the default) and without the use of query-dependent banding (QDB).

just running the full model (see QDB entry for DsrA). A method for selectively using only some of the partial models on each portion of the database is the only way to get a significant reduction in execution time and such a method is presented in the next section.

Since the U4 family is to be analyzed in detail, the specific partition used in the results to follow is shown in Fig. 4. This partition removes three bifurcations from the original model. It was chosen to generate five partitions of somewhat equal size. No attempt has been made to optimize the choice of partition boundaries or number of partitions. The automation of model partitioning has been left to future work.

4 Decision Trees

The strategy employed here to make efficient use of the partial models is to design a binary decision tree. Each nonleaf node of the tree specifies one of the partial models and a threshold value for the score generated by that partial model. The leaf nodes specify either reject or accept. Those locations in the database which reach an *Accept* node are then processed with the full covariance model such that the

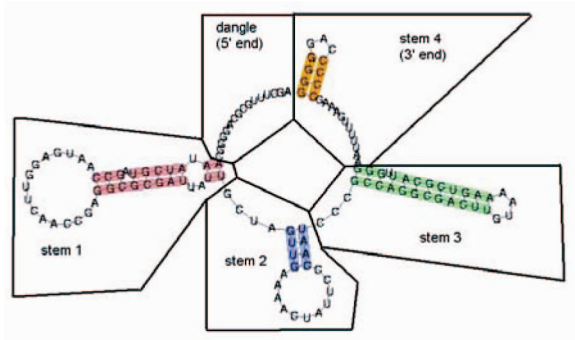

Fig. 4. U4 (RF00015) consensus secondary structure and sequence divided into five parts. On the $5'$ (left) end is a dangling end composed of all L nodes. The other four parts (stems 1-4) are each composed of a hairpin and connector regions.

false alarm rate will be no higher than if the whole database was processed with the full model. If the P-values at the thresholds chosen for each partial model are known, then the probability of reaching any given node in the decision can be calculated. It will be assumed that the fraction of the database containing true family members is insignificant such that the probability of reaching a decision tree node is nearly equal to the probability of reaching that node from a random portion of the database that does not contain a family member. The probabilities of reaching nodes can then be multiplied by the execution time associated with the model run at that node to find overall processing cost.

Fig. 5 shows a possible decision tree for the U4 family. Partial covariance model 1 is run on the entire database. This incurs a computation cost of 0.042 since the root node probability is 1.000 and the normalized cost of partial model 1 is 0.042 (from Table 1). Partial model 3 is then always run, but with two different thresholds depending on the results from running partial model 1. Since model 3 is always run (probabilities 0.835 plus 0.165 for the two child nodes of the root), the cost of these two nodes is 0.091 (again, see Table 1). The total cost can be found from adding up the

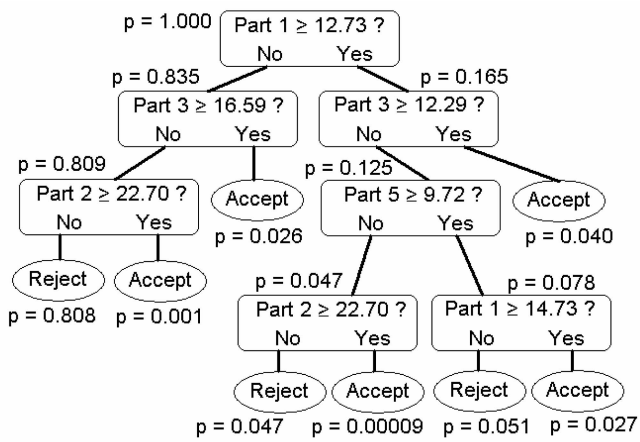

Fig. 5. A decision tree for the U4 (RF00015) RNA family using five partial covariance models. The probabilities p of reaching each node are shown. It is possible for the same partial model to appear again further down the tree (for example, Part 1 at both the tree root and in the lower right in this tree) as long as the lower model has a higher threshold than the upper. There is no additional computation cost for the second partial model since the partial model score has already been calculated once.

products of the node probabilities and the partial model normalized execution times where *Accept* nodes require full model execution. Using the decision tree in Fig. 5 and the five U4 partial models results in a mean execution time of 0.249 with QDB relative to using the full model alone.

The goal is to construct a decision tree with minimal average execution time subject to the constraint that all seed sequences in the family are accepted. Even with only four or five partial models, the number of possible combinations of tree topologies and thresholds is huge. There is no obvious algorithm to construct an optimal tree, so the use of computational intelligence methods is suggested.

One such method uses an evolutionary algorithm where each gene consists of a node type (a partial model number or terminal) and a threshold index. The fitness of a solution is one plus the fraction of seed sequences that lead to a *Reject* node if any of the seed sequences leads to a *Reject* node. Otherwise, the fitness is the average execution time. This fitness function should be minimized (or its negative should be maximized). If the best fitness that can be found is greater than 1, then it is better to revert to simply using the full covariance model. Since execution time is minimized subject to the constraint that all seed sequences pass the decision tree, there is never any reason to choose a partial model threshold different than one of the members of the set of scores of the seed sequences for the given partial model. The threshold index can then be a value in the range [1 S], where S is the number of seed sequences. The threshold used is the score of the indexed sequence against the selected partial model. It is often the case that several of the seed sequences are identical within the alignment column range of the partial model, and therefore, the partial model scores are identical for these sequences. This is actually an advantage, since the probability of choosing a particular threshold value is proportional to the number of sequences that use it. As an example, the U4 family has 30 seed sequences and has been broken into five partial models, so each genetic algorithm gene would have a value in the range [0 5] to specify node type (0 = terminal, m = partial model for $m > 0$) and a value in the range [1 30] for a threshold index.

Fig. 6 shows an example of a 17-gene chromosome which maps to the decision tree in Fig. 5. To convert an ordered list of genes into a decision tree, the first gene is the tree root, the second gene is the root's left child, the third gene is the root's right child, the fourth gene is the left child of the root's left child, etc. Some of the genes in the representation may be inactive since a higher level node's gene has a node type of *Reject* or *Accept* (these inactive genes are marked with X in Fig. 6). Nodes marked as terminals (node type 0) are taken to be *Reject* if they are a left child and *Accept* if they are a right child (for example, gene E has node type 0 and specifies an *Accept* node since it is the right child of gene B). Nodes which are not terminal might not have any children specified if the child chromosome positions are beyond the end of the chromosome. In this case, the nonterminal node is given Reject and Accept children as left and right child, respectively, even though these nodes have no associated gene. Partial model scores for each seed sequence (as in Table 2) are used to convert the threshold indexes into

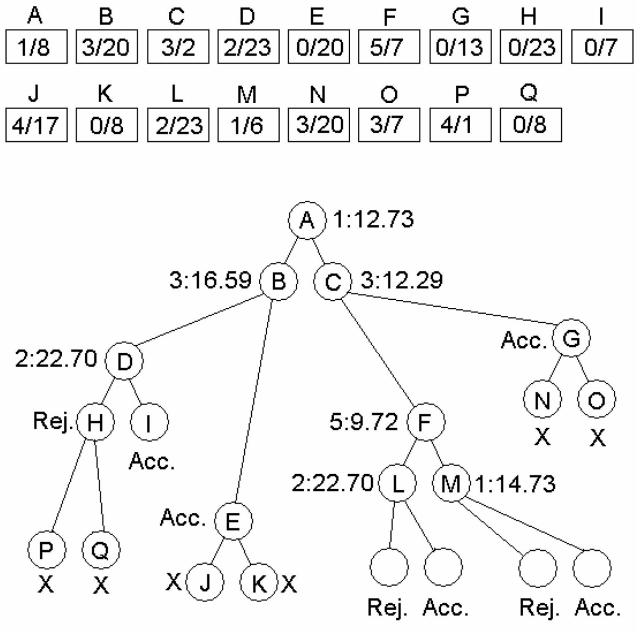

Fig. 6. Example genetic algorithm representation of the decision tree in Fig. 5. The chromosome has 17 genes (A-Q), each with at node type in the range [0 5] and a threshold index in range [1 30] given as type/index. A node type of 0 indicates a terminal node, which is a *Reject* node if it is a left child and an *Accept* node if it is a right child. Nodes marked with X are inactive since a node higher in the tree is a terminal *Reject* or *Accept* node. Extra *Reject* and *Accept* nodes are added as the left and right children, respectively, for nonterminal nodes whose children are beyond the end of the chromosome (for example, the children of L and M). The threshold indexes are used to obtain a threshold value using Table 2 (for example, gene A contains 1/8, which uses the partial model 1 score for the eighth sequence = 12.73).

threshold values. In order to avoid chromosomes getting excessively long, the chromosome may be pruned at the point where no genes to the right are active. Mutations take the form of selecting a new value for the node type and/or threshold index. Insertions, deletions, and crossover should be done on a whole gene basis.

5 Experimental Results

In this section, detailed results of using partial covariance models are shown for the U4 RNA family [11], [12], [13], [14] and summary results are presented for the MicC [15], [16], DsrA [17], [18], [19], HgcF [20], P1 [21], RyhB, and SL1 families. U4 is chosen because it is representative of RNA families with relatively long sequences and many known family members exhibiting groups of sequences with differing local sequence conservation. The other six families come from randomly selecting from the families with long consensus sequences and throwing out those which where too closely related. For example, the HgcA, HgcB, ..., HgcG, HhcA, and HhcB families are all "high GC" content ncRNA genes in thermophilic bacteria (although only HgcC, HgcE, HgcF, and HgcG currently have Rfam families), so only one of this class of families was considered for examination. The P1 sRNA gene in bacteria also has related genes P9, P11, P15, P16, P24, and P26 in Rfam. DsrA and MicC are found mostly in bacteria (except for one member each in a virus). The MicC sRNA possibly regulates the

TABLE 2
Full and Partial Model Scores for U4 Seed Sequences

Sequence	Full	Part 1	Part 2	Part 3	Part 4	Part 5	Sum
X58844	79.74	7.29	39.10	18.44	5.52	15.92	86.27
AF326336	47.28	14.73	8.26	12.29	1.87	14.76	51.91
AY007789	36.76	14.73	4.50	2.45	2.26	17.47	41.41
X97621	42.91	14.73	9.37	2.45	3.31	17.47	47.33
AJ245951	51.17	14.73	8.26	13.70	0.62	17.66	54.97
AF204671	50.86	14.73	3.08	17.70	2.48	18.61	56.60
M25777	38.09	14.73	12.06	8.44	1.14	9.72	46.09
U18778	45.03	12.73	28.56	12.35	5.74	6.23	65.61
X15491	70.52	16.07	23.83	18.67	4.38	5.89	68.84
K03095	88.95	16.04	38.40	21.95	11.35	5.38	93.12
AE003669	102.29	18.12	38.87	20.28	14.99	16.53	108.79
AE003664	103.36	18.12	39.37	20.28	14.99	16.53	109.29
K00474	109.73	17.75	38.72	24.71	17.75	19.99	118.92
M14136(1)	111.54	17.75	39.21	23.54	19.63	20.28	120.41
M14136(2)	112.24	17.75	38.72	24.71	19.63	19.99	120.80
M18004	110.38	17.75	39.21	24.71	17.75	19.99	119.41
K00782	91.86	8.96	39.21	24.71	18.55	10.92	102.35
AC004263	112.89	17.75	39.21	24.71	19.63	19.99	121.29
X59361	106.44	17.75	34.84	24.71	17.75	19.99	115.04
L22250	90.98	17.75	35.73	16.59	4.39	21.60	96.06
X51382	93.78	16.68	29.83	22.49	13.69	16.70	99.39
X07828	103.98	16.68	35.67	22.49	13.69	20.52	109.05
X13840	49.67	3.90	22.70	18.29	-1.20	8.06	51.75
X15933	104.48	13.83	36.07	19.34	18.62	9.97	97.83
X15932	104.34	13.83	33.25	19.34	16.98	14.11	97.51
X15931	97.13	13.83	34.69	19.34	16.44	12.12	96.42
X07112	96.71	13.83	35.00	18.33	16.65	11.61	95.42
X07113	103.40	13.83	35.00	17.46	22.85	12.15	101.29
X67146	96.07	13.83	35.21	19.34	14.66	12.12	95.16
X67145	100.84	13.83	36.59	19.34	17.64	11.58	98.98
Mean	85.11	14.80	29.75	18.44	11.79	14.80	89.58
Min	36.76	3.90	3.08	2.45	-1.20	5.38	41.41
Max	112.89	18.12	39.37	24.71	22.85	21.60	121.29

All scores are generated using the `cmbuild` *and* `cmsearch` *programs in the Infernal 0.81 suite with default settings (which includes using QDB in* `cmsearch`*). Scores are 1og (base 2) likelihood ratios with units of bits.*

TABLE 3
Full and Partial Model P-Values for U4 Seed Sequences

Sequence	Full	Part 1	Part 2	Part 3	Part 4	Part 5	Product
X58844	9.2E-12	9.8E-1	1.7E-6	1.2E-2	8.4E-1	4.0E-2	6.9E-10
AF326336	1.3E-07	5.7E-2	5.8E-1	2.4E-1	1.000	7.1E-2	5.7E-04
AY007789	3.0E-06	5.7E-2	9.9E-1	1.000	1.000	1.8E-2	1.0E-03
X97621	4.8E-07	5.7E-2	4.2E-1	1.000	0.999	1.8E-2	4.3E-04
AJ245951	4.2E-08	5.7E-2	5.8E-1	1.3E-1	1.000	1.7E-2	7.0E-05
AF204671	4.6E-08	5.7E-2	1.000	1.8E-2	1.000	1.0E-2	1.0E-05
M25777	2.0E-06	5.7E-2	1.6E-1	8.6E-1	1.000	6.2E-1	4.8E-03
U18778	2.6E-07	1.7E-1	1.5E-4	2.4E-1	7.9E-1	0.997	4.7E-06
X15491	1.4E-10	2.7E-2	1.2E-3	1.1E-2	9.8E-1	0.999	3.3E-07
K03095	6.1E-13	2.8E-2	2.3E-6	2.1E-3	4.4E-2	1.000	5.9E-12
AE003669	1.2E-14	8.6E-3	1.9E-6	4.8E-3	4.5E-3	2.9E-2	1.1E-14
AE003664	8.7E-15	8.6E-3	1.5E-6	4.8E-3	4.5E-3	2.9E-2	8.5E-15
K00474	1.3E-15	1.1E-2	2.0E-6	5.2E-4	7.9E-4	5.1E-3	4.5E-17
M14136(1)	7.8E-16	1.1E-2	1.7E-6	9.4E-4	2.4E-4	4.4E-3	1.7E-17
M14136(2)	6.3E-16	1.1E-2	2.0E-6	5.2E-4	2.4E-4	5.1E-3	1.4E-17
M18004	1.1E-15	1.1E-2	1.7E-6	5.2E-4	7.9E-4	5.1E-3	3.6E-17
K00782	2.6E-13	7.8E-1	1.7E-6	5.2E-4	4.8E-4	4.1E-1	1.3E-13
AC004263	5.2E-16	1.1E-2	1.7E-6	5.2E-4	2.4E-4	5.1E-3	1.1E-17
X59361	3.5E-15	1.1E-2	1.1E-5	5.2E-4	7.9E-4	5.1E-3	2.3E-16
L22250	3.3E-13	1.1E-2	7.3E-6	3.1E-2	9.8E-1	2.2E-3	5.2E-12
X51382	1.5E-13	1.9E-2	9.0E-5	1.6E-3	1.0E-2	2.7E-2	7.6E-13
X07828	7.2E-15	1.9E-2	7.5E-6	1.6E-3	1.0E-2	3.9E-3	9.1E-15
X13840	6.6E-08	1.000	1.9E-3	1.3E-2	1.000	9.0E-1	2.2E-05
X15933	6.2E-15	9.3E-2	6.3E-6	7.8E-3	4.6E-4	5.8E-1	1.2E-12
X15932	6.5E-15	9.3E-2	2.1E-5	7.8E-3	1.3E-3	9.8E-2	1.9E-12
X15931	5.5E-14	9.3E-2	1.1E-5	7.8E-3	1.8E-3	2.5E-1	3.7E-12
X07112	6.2E-14	9.3E-2	9.9E-6	1.3E-2	1.6E-3	3.1E-1	5.8E-12
X07113	8.6E-15	9.3E-2	9.9E-6	2.0E-2	3.1E-5	2.4E-1	1.4E-13
X67146	7.5E-14	9.3E-2	9.1E-6	7.8E-3	5.6E-3	2.5E-1	9.0E-12
X67145	1.8E-14	9.3E-2	5.0E-6	7.8E-3	8.5E-4	3.1E-1	9.7E-13
Log Mean	1.9E-12	4.6E-2	7.9E-5	9.3E-3	1.4E-2	5.4E-2	2.6E-11
Min	5.2E-16	8.6E-3	1.5E-6	5.2E-4	3.1E-5	2.2E-3	1.1E-17
Max	3.0E-06	1.000	1.000	1.000	1.000	1.000	4.8E-03

All P-values are generated using the `cmbuild` *and* `cmsearch` *programs in the Infernal 0.81 suite with default settings (which includes using QDB in* `cmsearch`*). P-values are based on fitting 1,000 random sequences to an extreme value distribution.*

OmpC protein and has a related family MicF which possibly regulates OmpF. U4 appears in many eukaryotes and is associated with other ncRNA in the spliceosome (such as U1, U2, U5, U6, U11, and U12). The selection of five RNA families is hopefully diverse enough to be indicative of RNA families with long consensus sequences. The average lengths of the five families are 137.9 (U4), 121.6 (MicC), 85.8 (DsrA), 168.6 (HgcF), 180.0 (P1), 65.0 (RyhB), and 103.0 (SL1). About three quarters of the 607 Rfam version 8.1 (October 2007) families have average length over 75 bases which was the cutoff for the definition of long used here.

Table 2 shows the scores generated by the full covariance model and each of the five partial models on all of the 30 seed sequences of the U4 RNA family in Rfam. The score threshold used in the Rfam 8.1 database is about 48.5, so some of the seed sequences (numbers 2, 3, 4, 7, and 8) would not be found in the database search if they were not already part of the hand curated seed family. The last column of the table shows the sum across all five partial models for each sequence and this value is generally larger than that of the full model. This is due to no penalty being applied to insertions between the models or database overlaps of the models (which would require deletions for the full model). However, the full model scores and sum of partial model scores are highly correlated as should be expected.

None of the seed sequences is completely identical to any other seed sequence, so none of the full model scores are identical to any other. It is not unusual that the portion of the sequences covered by a partial model is identical in one or more seed sequences. For example, in sequences 2 through 7, partial model 1 gives a score of 14.73 in every case since the sequences are identical from consensus position 1 through 16 in these sequences.

The minimum partial model scores near the bottom of Table 2 show that no single partial model does well on all 30 seed sequences. The maximum partial model scores show that all partial models do well on at least one of the sequences. Study of the scores for individual sequences indicate that some sequences might be detected with a high score on a single partial model, another sequence with a high score on a different partial model, and a third sequence might require moderate scores on more than one partial model. These complex relationships make it hard to find partial model application order and thresholds.

The P-values (probability that a random sequence with the same A, C, G, and U proportions as the consensus sequence will have a score at least as high as the true family member sequence) are shown in Table 3 for the 30 seed sequences. The products of the P-values across all five

TABLE 4
Decision Trees and Statistics for All Five RNA Families

Family	Decision Tree / Statistics
U4	(1:12.73, (3:16.59, (2:22.70, R, A), A), ... (3:12.29, (5:9.72, (2:22.70, R, A), (1:14.73, R, A)), A) Execution Time = 0.249 All Sequence Coverage = 96%
MicC	(2:9.98, R, (1:24.98, R, (3:27.08, R, (4:45.68, R, A)))) Execution Time = 0.156 All Sequence Coverage = 100%
DsrA	(1:26.10, R, (3:15.62, R, (2:25.64, R, A))) Execution Time = 0.268 All Sequence Coverage = 100%
HgcF	(1:14.06, R, (3:46.87, R, (2:60.46, R, A))) Execution Time = 0.145 All Sequence Coverage = 100%
P1	(1:15.16, R, (4:13.29, (3:29.82, R, A), A) Execution Time = 0.066 All Sequence Coverage = 100%
RyhB	(2:27.42, R, (3:21.97, R, (1:17.82, R, A))) Execution Time = 0.162 All Sequence Coverage = 95%
SL1	(3:26.35, R, (1:15.43, R, A)) Execution Time = 0.203 All Sequence Coverage = 100%

The notation (Part:Thesh, No, Yes) indicates that partial model Part is run with threshold Thresh with left child node No and right child node Yes. R and A refer to Reject and Accept nodes, respectively. Fig. 5 shows a graphical expansion for U4.

TABLE 5
Mean, Minimum, and Maximum Partial Scores for Five Families

Family	Full	Part 1	Part 2	Part 3	Part 4	Part 5
U4						
Mean	85.11	14.80	29.75	18.44	11.79	14.80
Min	36.76	3.90	3.08	2.45	-1.20	5.38
Max	112.89	18.12	39.37	24.71	22.85	21.60
MicC						
Mean	120.52	15.38	25.31	31.23	49.81	
Min	109.42	9.98	24.98	27.08	45.68	
Max	125.48	17.23	26.03	32.59	52.36	
DsrA						
Mean	82.15	27.74	30.68	24.30		
Min	70.54	26.10	25.64	18.54		
Max	92.41	28.73	35.58	29.09		
HgcF						
Mean	200.16	34.57	77.98	87.67		
Min	194.68	31.68	76.85	86.07		
Max	202.90	36.02	78.54	88.47		
P1						
Mean	144.22	20.60	62.59	32.76	13.85	
Min	124.02	15.16	49.51	29.82	9.85	
Max	159.17	24.87	78.09	39.69	20.44	
RyhB						
Mean	91.57	22.38	31.49	34.73		
Min	72.45	16.33	24.92	20.41		
Max	101.76	32.52	47.36	39.45		
SL1						
Mean	98.22	27.88	26.48	41.11		
Min	69.22	19.17	15.43	26.35		
Max	107.51	36.83	40.44	52.76		

partial models is shown in the last column and is generally larger than that of the full model. This indicates that the false alarm rate when using all five partial models is more than when using the full model. In a number of cases, this false alarm rate is unacceptably high. Since Rfam uses a threshold of about 48.5, a P-value of less than about 1E-7 is reasonable (AF326336 has a score of 47.28 and P-value of 1.3E-7, and AF204671 has a score of 50.86 and a P-value of 4.6E-8). Sequences 2, 3, 4, 7, and 8 do not meet the Rfam threshold, so P-values of more than 1E-7 using partial models are not a great problem, but other sequences such as numbers 5, 6, 9, and 23 have P-values using all five partial models that are too high. If not all partial models are to be run, there are other sequences that will cause unacceptably high P-values if thresholds are set low enough to find them. Clearly, it is necessary to run the full model on the accepted portion of the database using the partial models in order to get acceptable false alarm rates.

Decision trees have been found for each of the seven RNA families and are shown in Table 4 along with average normalized execution time and fraction of all known sequences passed to the full model for scoring. Each node of the decision tree is represented by three items surrounded by parentheses. The first item is a partial model number with threshold, the second item is the left child node (child node for case where the partial model score does not exceed threshold), and the third item is the right child node (child node for case where the partial model score exceeds threshold). R and A refer to *Reject* and *Accept* nodes, respectively, in this notation. The entry for U4 in Table 4 may be compared to Fig. 5, which graphically shows the same tree. The average normalized execution times range from 0.066 to 0.268 of the execution time required for full-model search. For U4, the estimated time to run the full CM on an eight gigabase database using a single 2.8 GHz Pentium 4 is 1,258 days [8], so a normalized runtime of 0.249 implies a runtime of 313 days instead. In all cases, the fraction of known RNA family members found is 95 percent or better.

One notices from the decisions trees in Table 4 that the trees for the four non-U4 cases are considerably simpler. This is a result of not having subfamily diversity within the overall RNA family. In U4 (and some other Rfam families), the overall family could have been divided up into separate families. The choice of what constitutes a family involves a trade-off between better parameter estimation from having more training sequences and loss of detail by averaging across groups of sequences with features that are similar within group but heterogeneous between groups. In Table 5, one sees that the four non-U4 families have considerably less variation in scores for each partial model. If one examines the sequences of these families, there are no apparent subfamily groups that might potentially be considered families of their own.

Decision trees for these simpler families without subfamily diversity are easier to generate without the use of computational intelligence tools. If the least computationally expensive partial model combined with a threshold equal to the score of the lowest scoring seed sequence has P-value less than the ratio of its execution time to the next smallest execution time, then this partial model and threshold combination makes a very good root node in the decision tree. The left child of the root in such case should be a *Reject*

node. This rule can be iterated on the right child of the root and subsequent right children. This is exactly the situation in the MicC, DsrA, and HgcF trees shown. In P1, the root node also follows this rule. Recognizing these cases can reduce the need to apply computational intelligence tools where they are not really necessary. However, U4 is not the only family with a long consensus sequence and a large number of known family members that might potentially have a complex decision tree.

Table 5 also illustrates a problem with making partial models too small. In general, the variability in partial model scores relative to the mean score is greater than that of the full model. As the full model gets divided up into increasing smaller partial models, this tendency would become even more noticeable since smaller multiple alignment column ranges are more likely to be completely covered by a region of nonconservation in one or more of the member sequences. So, very small partial models will have good execution times, but are unlikely to have a P-value on the lowest scoring seed sequence such that a *Reject* node left child is desirable. With many highly erratic small models to choose from, the potential to overfit the decision tree to the seed sequences also increases. A very short segment of the multiple alignment which just happens to be nearly identical for all the seed sequences might not be so well conserved in yet undiscovered family members. As a result, division of a full model into three to five partial models appears to be a useful approach, but the usefulness of dividing into 10 or more partial models is questionable (unless the consensus sequence is extremely long).

6 Conclusion

It has been shown that the technique of reducing computation time for covariance-model-based database search for RNA family members using partial covariance models is potentially useful. In some cases, the choice of partial model application order and threshold values can be quite complex (as with the U4 RNA family) and may require some form of automated search to obtain a good choice. Binary decision trees allow a reasonable way to describe the partial model order and threshold choices. Computational intelligence methods can be applied to find good decision trees without too much difficulty. For some cases, these computational intelligence methods are unnecessary since the decision tree is relatively obvious. These simple cases are generally associated with RNA families that do not have subfamilies that might reasonably be families in their own right.

Execution times in the range 0.066-0.268 of the full model have been shown for seven RNA families with no more than 5 percent loss in family sequences found (and no loss at all in five out of seven families). Execution time reductions could potentially be even larger since the number of partial models and their boundaries were chosen in an ad hoc fashion and there is no guarantee that the decision trees used are optimal. Future work on this topic would include investigation of the best method to select partial models, including the possibility of overlapping partial models or a partial model set that does not cover every multiple alignment column. The possibility that base pairs that were ignored in the original full model to avoid having pseudoknots [22] might be included in overlapping partial models is also open to investigation.

Acknowledgments

This work was supported in part by the US National Institutes of Health under Grant P20 RR016454 from the INBRE Program of the National Center for Research Resources.

References

[1] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge Univ. Press, 1998.
[2] S. Eddy, "Computational Genomics of Noncoding RNA Genes," *Cell,* vol. 109, pp. 137-140, 2002.
[3] S. Eddy and R. Durbin, "RNA Sequence Analysis Using Covariance Models," *Nucleic Acids Research,* vol. 22, pp. 2079-2088, 1994.
[4] S. Eddy, "Hidden Markov models," *Current Opinion in Structural Biology,* vol. 6, pp. 361-365, 1996.
[5] S. Eddy, Infernal User's Guide (version 0.81), http://infernal.janelia.org, May 2007.
[6] S. Griffiths-Jones, A. Bateman, M. Marshall, A. Khanna, and S. Eddy, "Rfam: An RNA Family Database," *Nucleic Acids Research,* vol. 31, pp. 3608-3612, 2003.
[7] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman, "Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs," *Nucleic Acids Research,* vol. 25, pp. 3389-3402, 1997.
[8] Z. Weinberg and W. Ruzzo, "Faster Genome Annotation of Non-Coding RNA Families without Loss of Accuracy," *Proc. Eighth Ann. Int'l Conf. Research in Computational Molecular Biology,* pp. 243-251, 2004.
[9] Z. Weinberg and W. Ruzzo, "Sequence-Based Heuristics for Faster Annotation of Non-Coding RNA Families," *Bioinformatics,* vol. 22, pp. 35-39, 2006.
[10] E. Nawrocki and S. Eddy, "Query-Dependent Banding (QDB) for Faster RNA Similarity Searches," *PLoS Computational Biology,* vol. 3, pp. 540-554, 2007.
[11] C. Zwieb, "The uRNA Database," *Nucleic Acids Research,* vol. 25, pp. 102-103, 1997.
[12] I. Vidovic, S. Nottrott, K. Hartmuth, R. Luhmann, and R. Ficner, "Crystal Structure of the Spliceosomal 15.5 kD Protein Bound to a U4 snRNA Fragment," *Molecular Cell,* vol. 6, pp. 1331-1342, 2000.
[13] P. Raghunathan and C. Guthrie, "A Spliceosomal Recycling Factor that Reanneals U4 and U6 Small Nuclear Ribonucleoprotein Particles," *Science,* vol. 279, pp. 857-860, 1998.
[14] J. Thomas, K. Lea, E. Zucker-Aprison, and T. Blumenthal, "The Spliceosomal snRNAs of Caenorhabditis Elegans," *Nucleic Acids Research,* vol. 18, pp. 2633-2642, 1990.
[15] S. Chen, E. Lesnik, T. Hall, R. Sampath, R. Griffey, D. Ecker, and L. Blyn, "A Bioinformatics Based Approach to Discover Small RNA Genes in the Escherichia coli Genome," *Biosystems,* vol. 65, pp. 157-177, 2002.
[16] S. Chen, A. Zhang, L. Blyn, and G. Storz, "MicC, A Second Small-RNA Regulator of Omp Protein Expression in Escherichia coli," *J. Bacteriology,* vol. 186, pp. 6689-6697, 2004.
[17] D. Sledjeski and S. Gottesman, "A Small RNA Acts as an Antisilencer of the H-NS-Silenced rcsA Gene of Escherichia coli," *Proc. Nat'l Academy of Science USA,* vol. 92, pp. 2003-2007, 1995.
[18] N. Majdalani, C. Cunning, D. Sledjeski, T Elliot, and S. Gottesman, "DsrA RNA Regulates Translation of RpoS Message by an Anti-Sense Mechanism, Independent of Its Action as an Antisilencer of Transcription," *Proc. Nat'l Academy of Sciences USA,* vol. 95, pp. 12462-12467, 1998.
[19] S. Gottesman, "Micros for Microbes: Non-Coding Regulatory RNAs in Bacteria," *Trends in Genetics,* vol. 21, pp. 399-404, 2005.
[20] R. Klein, Z. Misulovin, and S. Eddy, "Noncoding RNA Genes Identified in AT-Rich Hyperthermophiles," *Proc. Nat'l Academy of Sciences USA,* vol. 99, pp. 7542-7547, 2002.

[21] L. Livny, A. Brencic, S. Lory, and M. Waldor, "Identification of 17 Pseudomonas Aeruginosa sRNAs and Prediction of sRNA-Encoding Genes in 10 Diverse Pathogens Using the Bioinformatic Tool sRNApredict2," *Nucleic Acids Research,* vol. 34, pp. 3484-3493, 2006.
[22] D. Staple and S. Butcher, "Pseudoknots: RNA Structures with Diverse Functions," *PLoS Biology,* vol. 3, pp. e213, 2005.
[23] K. Lagesen, P. Hallin, E. Rødland, H. Stærfeldt, T. Rognes, and D. Ussery, "RNAmmer: Consistent and Rapid Annotation of Ribosomal RNA Genes," *Nucleic Acids Research,* vol. 35, pp. 3100-3108, 2007.
[24] Z. Weinberg, "Accurate Annotation of Non-Coding RNAs in Practical time," PhD thesis, Univ. of Washington, 2005.
[25] R. Klein and S. Eddy, "RSEARCH: Finding Homologs of Single Structured RNA Sequences," *BMC Bioinformatics,* vol. 4, p. 44, 2003.

Jennifer A. Smith received the PhD degree in electrical engineering from the University of Idaho in 2003 and the MS degree in electrical engineering from the University of Connecticut in 1993. She is an assistant professor of computer engineering at Boise State University in Idaho. Her research interests are in noncoding RNA gene search, particularly in acceleration of search through algorithm redesign and special purpose computing hardware. She has been active with the IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology since its inception and most recently served as the Symposium Chair for CIBCB 2008. She is a senior member of the IEEE.