7

Covariance-Model-Based RNA Gene Finding: Using Dynamic Programming versus Evolutionary Computing

Scott F. Smith

Dept. of Electrical and Computer Engineering, Boise State University, Boise, Idaho, 83725-2075, USA
sfsmith@boisestate.edu

Summary. This chapter compares the traditional dynamic programming RNA gene finding methodolgy with an alternative evolutionary computation approach. Both methods take a set of estimated covariance model parameters for a non-coding RNA family as given. The difference lies in how the score of a database position with respect to the covariance model is computed. Dynamic programming returns an exact score at the cost of very large computational resource usage. Presently, databases are prefiltered using non-structural algorithms such as BLAST in order to make dynamic programming search feasible. The evolutionary computing approach allows for faster approximate search, but uses the RNA secondary structure information in the covariance model from the start.

7.1 Introduction

The initial focus of interpreting the output of sequencing projects such as the Human Genome Project [1] has been on annotating those portions of the genome sequences that code for proteins. More recently, it has been recognized that many significant regulatory and catalytic functions can be attributed to RNA transcripts that are never translated into protein products [2]. These functional RNA (fRNA) or non-coding RNA (ncRNA) molecules have genes which require an entirely different approach to gene search than protein-coding genes.

Protein-coding genes are usually detected by gene finding algorithms that generically search for putative gene locations and then later classify these genes into families. As an example, putative protein-coding genes could be identified using the GENESCAN program [3]. Classification of these putative protein-coding genes could then be done using profile hidden Markov models (HMMs) [4] to yield families of proteins (or protein domains) such as that in Pfam [5]. It is not necessary to scan entire genomes with an HMM since a small subset of the genome has already been identified by the gene finding algorithm as possible protein-coding gene locations. Unlike protein-coding genes, RNA genes are not associated with promoter regions and open reading frames. As a result, direct search for RNA genes using only

S.F. Smith: *Covariance-Model-Based RNA Gene Finding: Using Dynamic Programming versus Evolutionary Computing*, Studies in Computational Intelligence (SCI) **94**, 183–208 (2008)
www.springerlink.com

generic characteristics has not been successful [6]. Instead, a combined RNA gene finding and gene family classification is undertaken using models of a gene family for database search over entire genomes. This has the disadvantage that RNA genes belonging to entirely novel families will not be found, but it is the only currently available method that works. It also means that the amount of genetic information that needs to be processed by the combined gene finder and classifier is much larger than for protein classifiers.

Functional RNA is made of single-stranded RNA with intramolecular base pairing. Whereas protein-coding RNA transcripts (mRNA) are primarily information carriers, functional RNA often depends on its three dimensional shape for the performance of its task. This results in conservation of three dimensional structure, but not necessarily primary sequence. The three dimensional shape of an RNA molecule is almost entirely determined by the intramolecular base pairing pattern of the molecule's nucleotides. There are many examples of RNA families with very little primary sequence homology, but very well conserved secondary structure (see pp. 264–265 in [7]). It is very difficult to find RNA genes without taking conservation of secondary structure into account.

Most homology search algorithms such as BLAST [8], Fasta [9], Smith-Waterman [10], and profile HMMs only model primary sequence and are therefore not well suited for RNA gene search. These algorithms are in the class of regular grammars in the Chomsky hierarchy of transformational grammars [11]. In order to capture the long-range interactions embodied in RNA secondary structure, one needs to move up one level in the Chomsky hierarchy to a context-free grammar. The extension of the regular-grammar-based HMM to a context-free grammar is a covariance model (CM) [12].

The structure of covariance models and model parameter estimation from a secondary-structure-annotated multiple alignment of a RNA gene family is the subject of the next section. The use covariance models for gene search by a specific non-coding RNA database (Rfam) will be examined in Section 2. It will be seen that the traditional dynamic-programming method of scoring database locations with respect to a covariance model is so computationally intensive that filters are normally first used to reduce the amount of searched database by orders of magnitude. The advantages and drawbacks of these filters are discussed in Section 3. An alternative to filtering is introduced in Section 4, where an evolutionary-computation CM-based search method is shown. Finally, conclusions are drawn and discussion of work that remains to be done is undertaken.

7.2 Review of Covariance Models for RNA Gene Finding

Covariance models can be viewed as an extension of profile hidden Markov models such that covariation in nucleotides at model positions that are widely separated in sequence, but physically connected as base pairs is captured statistically. Profile hidden Markov models are a specific form of hidden Markov model in which state transitions have a unidirectional flow from the start (5' in RNA/DNA or N-terminal

in proteins) to the end (3' in RNA/DNA or C-terminal in proteins) of the model's consensus sequence. Similarly, a CM has unidirectional flow state transitions, but a more complicated connection topology. Profile hidden Markov models have five different types of states (start, match, insert, delete, and end). A CM has seven distinct state types (start, match pair, match/insert left, match/insert right, delete, bifurcate, and end). Finally, both types of models associate a group of states with each sequence position in the consensus sequence of the model. For the profile HMM, one match, one insert, and one delete state is associated with each consensus position (with possible exception of the first and/or last position). For the CM, a group of states (called a node) is associated with each consensus based-pair of positions and consensus unpaired position.

Both profile HMM and CM parameters are estimated from a group of nucleotide or protein sequences know as a family. In the case of the CM, it is also necessary to have a consensus secondary structure. This secondary structure may either be observed experimentally, or predicted from the sequence. In the case of non-coding RNA genes, prediction could be done with the Mfold [13] or RNAPredict [14] programs for example. The sequences may be either in the form of a multiple alignment or unaligned. For clarity of exposition, it is assumed here that the sequences are available in aligned form. In this case, the structure of the HMM or CM is determined by selecting alignment columns as either conserved or as insertion columns. Conserved columns are associated with some form of match state in the model. The most abundant symbol (nucleotide or amino acid) in each conserved column is taken as the consensus symbol for that position in the consensus sequence. The consensus sequence has length equal to the number of conserved multiple alignment columns.

7.2.1 CM Model Structure Determination

A multiple alignment that could be used to form a CM for the U12 family of non-coding RNA genes is shown in Figure 7.1. Included in the alignment are the seven sequences used by the Rfam database (described later) to form its CM of the family. These sequences are called seed sequences in Rfam and the resulting model has been used to find seven additional U12 family members. The four rows following the seven sequences contain consensus information. These four rows show consensus secondary structure, consensus sequence, CM node type assigned to the consensus column, and CM model branch letter code (used for reference to Figure 7.3 below) respectively. In the consensus structure rows, the symbol "-" indicates an unpaired conserved column, the symbols < and > represent the left (5'-side) and right (3'-side) halves of two base-paired conserved columns, and "." represents a non-conserved column. Note that there is no indication of which < column base pairs with which > column. This is because the structure is assumed to not have any pseudoknots. If the actual structure does have pseudoknots, then some of the base-paired columns have to be treated as if they were not base-paired. This results in some loss of power in the model, but the CM is not capable of representing pseudoknots and the increase in computational complexity of a model that can handle general pseudoknots is too high. If pseudoknots are disallowed, then the base-pairing notation is unambiguous.

Non-pseudoknotted structures have all base pairs such that all other base pairs are either completely inside or completely outside of them.

Of the six types of CM nodes, three do not emit consensus symbols (S = start, B = bifurcation, and E = end), two emit a single consensus symbol (L = left and R = right), and one emits two consensus symbols (P = pair). Therefore, only L, R, and P node types appear in the node type rows of Figure 7.1. L and R nodes are associated with a single multiple alignment column. Each P node is associated with two columns, one with secondary structure notation < and one with notation >. The multiple alignment shown has 156 columns, 149 of which represent consensus sequence positions. Four of the columns are assigned to R nodes, 63 to L nodes, and 41 pairs of columns to P nodes for a total of 4 + 63 + 2*41 = 149 consensus columns. Any time that a column could be assigned to either an L or an R node, the L node is preferred by convention, so there are generally many more L than R nodes in covariance models.

First, let us consider models that do not allow for insertions or deletions with respect to the consensus sequence. Such a profile HMM in shown in Figure 7.2 and a CM drawn at a level of detail that does not show insertion and deletion possibilities is shown in Figure 7.3. The HMM has only match (M) states. The arrow at the top of the figure shows that the model is written to process nucleotide sequences in the 5' to 3' direction. Each match state is associated with four emission probabilities, one for each of the four possible nucleotides. The nucleotide with highest probability is shown in parentheses inside the box for the M state and is equivalent to the consensus symbol for the conserved column of the multiple alignment represented by the M state. All transition probabilities is this model are equal to 1 since there is never any transition choice.

The node structure of the CM for the U12 RNA family is shown in Figure 7.3. The arrows pointing to the right and/or left of the R, L, and P nodes represent emission of a consensus symbol. This is a condensed version of the node structure where the numbers next to the emission arrows indicate that there is a vertical chain of that many nodes of identical type. The circled lower-case letters correspond to the branch codes in Figure 7.1. The first two consensus columns in Figure 7.1 (on the 5' end) correspond to two L nodes near the top of Figure 7.3 and the last three consensus columns in Figure 7.1 (on the 3' end) correspond to three R nodes just below in Figure 7.3. These two L nodes and three R nodes are all in branch "a". CM node diagrams always take the form of a binary tree. The top node in the tree is always an S node (called the root start node) and the bottoms of all terminal branches are always E nodes. Branches split into two at B nodes and the two children of any B node are always S nodes (called a left child start node and right child start node respectively).

Two of the branches in Figure 7.3 are surrounded by dashed boxes (branches a and b). All of branch a and the top portion of branch b are shown in expanded form in Figure 7.4. As shown by the arrows, the left side of a CM branch should be read from top to bottom and the right side of a branch from bottom to top in order to read consensus symbols in left-to-right multiple-alignment order (5' to 3' order). The a branch shows the first two consensus symbols are UG and the last three consensus symbols are CCG. The nodes actually represent a set of emission

```
1 .UGCCUUAAACUUAUGAGUAAGGAAAAUAACAACU......CGGGGUGAC
2 .UGCCUUAAACUUAUGAGUAAGGAAAAUAACGAUU......CGGGGUGAC
3 .UGCCUUAAACUUAUGAGUAAGGAAAAUAACGAUU......CGGGGUGAC
4 AUGUCUUAAACUUAUGAGUAAGGAAAAUAACGAUUGUUAUUCGGGGUGAU
5 .UGCCUUAAACUUAUGAGUAAGGAAAAUAACGAUU......CGGGGUGAC
6 AUGCCUUAAACUUAUGAGUAAGGAAAAUAACGAUU......CGGGGUGAC
7 AUGUCUUAAACUUAUGAGUAAGGAAAAUAACGAUUGUUAUUCGGGGUGAU
  .--<<<<<----------->>>>>--------<<<<.....<<<<-----
  .UGCCUUAAACUUAUGAGUAAGGAAAAUAACGAUU......CGGGGUGAC
  .LLPPPPPLLLLLLLLLLPPPPPLLLLLLLLPPPP......PPPPLLLLL
  .aabbbbbbbbbbbbbbbbbbbcccccccccdddd......ddddddddd

1 GCCCGAGUCCUCACUACUGAUGUGAGAGGAAUUUUUGUGCGGGUACAGGU
2 GCCCGAGUCCUCACUGCUUAUGUGAGAAGAAUUUUUGAGCGGGUAUAGGU
3 GCCCGAAUCCUCACUGCUAAUGUGAGACGAAUUUUUGAGCGGGUAAAGGU
4 GCCCGAAUCCUCACUGCUAAUGUGAGACGAAUUUUUGAGCUGGUAAAGGU
5 GCCCGAAUCCUCACUGCUAAUGUGAGACGAAUUUUUGAGCGGGUAAAGGU
6 GCCCGAAUCCUCACUGCUAAUGUGAGACGAAUUUUUGAGCGGGUAAAGGU
7 GCCCGAAUCCUCACUGCUAAUGUGAGACGAAUUUUUGAGCUGGUAAAGGU
  ->>>>>>>>>>><<<<<-------->>>>>----------<<<<<<<<<---<<<
  GCCCGAAUCCUCACUGCUAAUGUGAGACGAAUUUUUGAGCGGGUAAAGGU
  LPPPPPPPPPPPPPLLLLLLLLPPPPPLLLLLLLLLLLPPPPPPPLLLPPP
  ddddddddddeeeeeeeeeeeeeeeeffffffffffgggggggggggggg

1 CGUCCCC.GGGUGACCCGCUUACUUCGCGGGAUGCCCAGGUGCAAUGAUCUGCCCG
2 UGCAAUCUGAGCGACCCGCCUACUUUGCGGGAUGCCUGGGUGACGCGAUCUGCCCG
3 CGCCCUCAAGGUGACCCGCCUACUUUGCGGGAUGCC.....................
4 CGCCCCUAAGGUGACCAGCCUACUUUGCGGGAUGCCUAGGAGUCGCGAUCUGCCUG
5 CGCCCUCAAGGUGACCCGCCUACUUUGCGGGAUGCC.....................
6 CGCCCUCAAGGUGACCCGCCUACUUUGCGGGAUGCCUGGGAGUUGCGAUCUGCCCG
7 CGCCCCUAAGGUGACCAGCCUACUUUGCGGGAUGCCUAGGAGUCGCGAUCUGCCUG
  <<<<-----><<<<<<<->>>>>>>--<<<<<<<<<--------->>>>>>>>>---
  CGCCCUCAAGGUGACCCGCCUACUUUGCGGGAUGCCUAGGAGUCGCGAUCUGCCCG
  PPPPLLLLLPPPPPPPRPPPPPPPLLPPPPPPPPPLLLLLLLLLPPPPPPPPRRR
  gggggggggggggggggggggghhhhhhhhhhhhhhhhhhhhhhhhhhhaaa
```

Fig. 7.1. U12 multiple alignment of seven seed family members

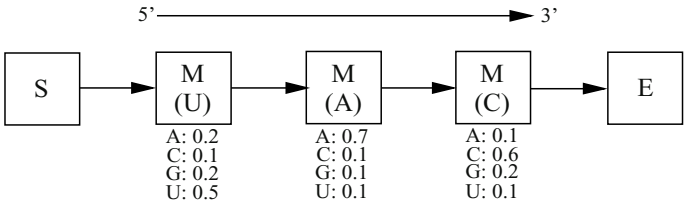

Fig. 7.2. A profile HMM with no insert or delete states

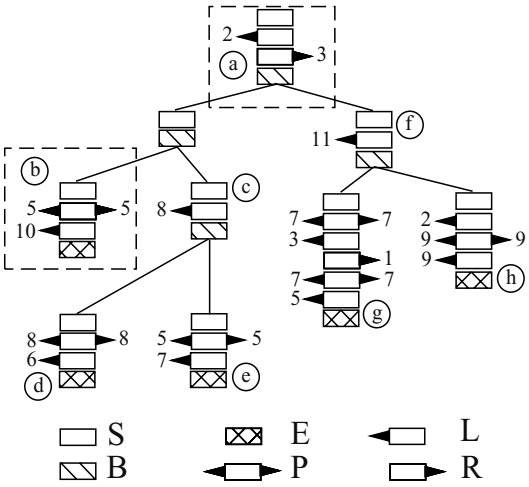

Fig. 7.3. Condensed U12 CM node structure

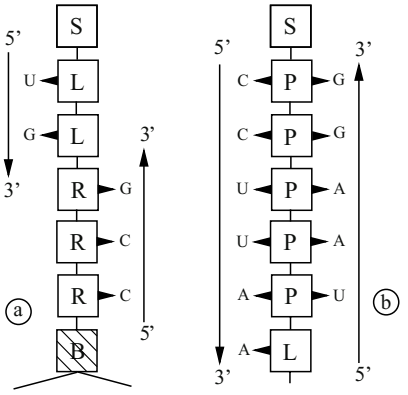

Fig. 7.4. Expanded portions of U12 CM node structure

probabilities. The L node directly below the root start node emits U with highest probability, by the probabilities of A, C, and G may be nonzero. L and R nodes have four such match probabilities (one for each possible nucleotide) and P nodes have sixteen probabilities (one for each possible pair of nucleotides). At the node structure level, the CM is similar to the HMM without insert or delete states in that transitions from a child node to a parent node happen with probability 1. At the bifurcations, the two submodels represented by the two subtrees are simply joined as one contiguous sequence with the left child subsequence on the left and the right child subsequence on the right.

Profile HMMs are normally augmented with insert and delete states as shown in Figure 7.5. The delete states (D states) are silent states that do not emit any symbols. These states simply allow one or more consensus positions in the model to be

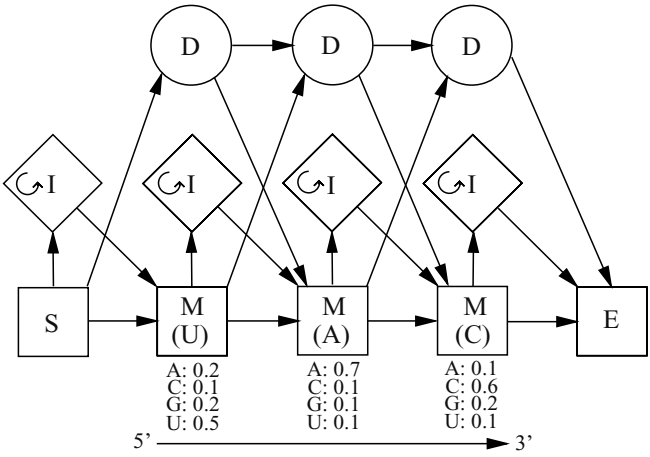

Fig. 7.5. A profile HMM with insert and delete states

skipped. The deletion penalty is imposed by the normally lower transition probabilities associated with the arrows to, from, and between D states as compared to the higher transition probabilities between M states. Affine delete gap penalties are available by having higher transition probabilities on D-to-D transitions than on M-to-D or D-to-M transitions. This allows multiple sequential deletions to be penalized less heavily than the same number of scattered deletions. This is consistent with observed gaps in nature and with gap penalties used in algorithms such as Smith-Waterman. The insert states (I states) have loop arrows inside the state diamond symbols to remind us that insert states always have self-loop transitions (both in HMMs and in CMs). This allows more than one possible insertion between consensus symbols. Affine insertion gap penalties are possible with differing self-loop and I-to-M/M-to-I transition probabilities. Unlike constant gap initiation and gap continuation penalties commonly used in algorithms such as Smith-Waterman, the gap penalties in an HMM or CM are position specific and can be different for insertions versus deletions. This leads to more flexibility, but also a large number of free parameters.

While it is possible to also include I-to-D and D-to-I transitions, Figure 7.5 omits these. Direct insertion to deletion transitions are rarely observed in real data and inclusion of these transitions just adds to the number of free parameters. The lack of these transitions is referred to as "plan seven" in the HMMER literature [15] (a program which estimates and scores profile HMMs). Seven refers to the number transitions leaving the D-I-M state triple associated with a consensus model position (including the I-state self loop). The alternative "plan nine" HMM architecture is not as commonly used. The standard CM is equivalent to a plan nine HMM in the sense that direct deletion to insertion transitions (and vice versa) are allowed. Investigation of the effect of removing these transitions in the CM case do not appear to have been undertaken to date.

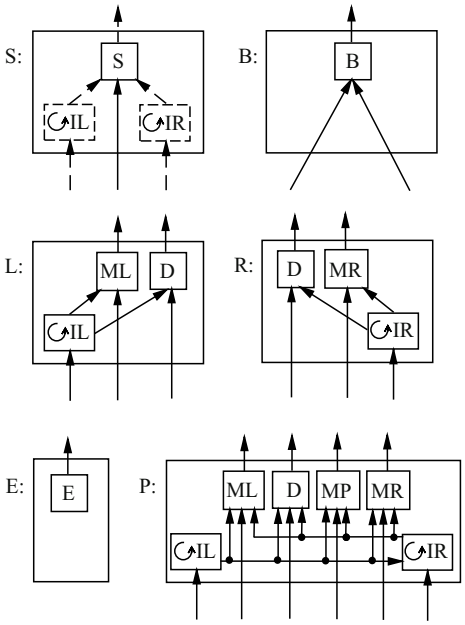

Fig. 7.6. Internal state structures of CM nodes

The equivalent to adding I and D states to the HMM is to allow non-consensus states within the nodes of the CM. Figure 7.6 shows the internal state structure of the six types of CM nodes. Each type of node contains a consensus state plus zero to five non-consensus states. The S, B, L, R, E, and P nodes have consensus states S, B, ML, MR, E, and MP respectively. Emitting nodes (L, R, and P) have D (delete) states that allow the consensus emitting state to be bypassed. The P node also contains two states that allow only the right or left half of the consensus pair to be missing in the database sequence. The ML state in the P node allows the right half of the consensus pair to be absent and the MR state allows the left half to be absent. IL and IR states allow additional database symbols to be inserted between consensus positions. These insert states have self-loop transitions, as indicated by the circular arrows inside the IL and IR state boxes such that any number of symbols may be inserted. The choice of which insert states to place in which node types is done such that there is no ambiguity as to which insert state in which node is responsible for insertions between a given couple of sequentially adjacent consensus locations. There are three sub-types of S nodes. The root S node has both IL and IR states. The right child S node has only an IL state. The left child S node has no insert states.

There are two levels of states within each node. The consensus state, D state (if present), and delete-related states (ML and MR in P nodes) are in the top level. The bottom level contains any insert states (IL or IR) that may be present. This implies that any insertions are added to the database sequence before any consensus matching is done (since the model is evaluated from the leaves toward the root). All top level

states in a given node have transitions to all states in the parent node. Bottom level states (insert states) only have transitions to top level states in the same node and to themselves. As a result, the arrows entering or leaving a node in Figure 7.6 represent a bundle of transitions whose number depends on the type of parent or child node. The IL- or IR- to-D transitions are clearly seen in Figure 7.6, but the D-to -IL or -IR transitions are only implicitly shown. These transitions make the standard CM architecture equivalent to a plan nine profile HMM.

7.2.2 Mapping a Database Sequence to a Covariance Model

In order to fit a database sequence to a covariance model one starts at the E states and works up the CM tree toward the root S state. Each E state models a null sequence and with no database symbols mapped to it. Transitioning from a child state to a parent state maps zero, one, or two database symbols to the model. Non-emitting parent states map no symbols, single-emitting states (IL, IR, ML, and MR) map a single symbol to the model, and the pair-emitting state (MP) maps two symbols. The transition adds a log-likelihood ratio transition score to the overall model score. If the parent state is an emitting state, a log-likelihood ratio emission score is also added.

Figure 7.7 shows the effect of moving from a top-level state T in a child node (of any type) to the top-level MR state in a parent R node. This has the effect of matching a database symbol to a model consensus symbol and inserting zero or more database symbols between the existing mapped database symbols and the consensus-matched symbol. If at least one database symbol is to be inserted, the first transition is from the child-node top-level state to the IR state of the parent R node. The length of the mapped sequence increases by one and the overall score increases by the T-to-IR transition score plus the IR state emission score for the inserted symbol (the emitted symbol is G in Figure 7.7). If an additional database symbol is inserted, the mapped sequence again increases in length by one and the overall score increases by the IR-to-IR transition score plus the IR state emission score of the new symbol. This continues until all inserted symbols are finished. Finally, the consensus-matched symbol (U in the figure) increases the mapped sequence length by one and increases the score by the IR-to-MR transition score plus the MR state emission score for the

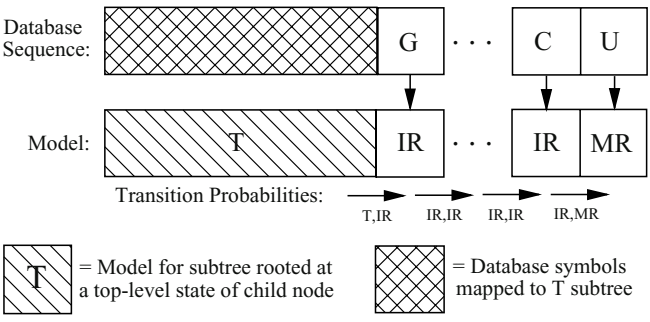

Fig. 7.7. Building to the right with an R node

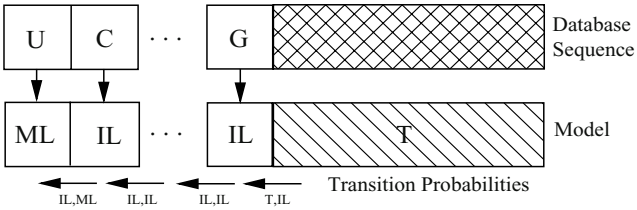

Fig. 7.8. Building to the Left with an L node

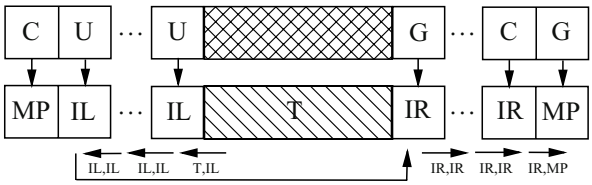

Fig. 7.9. Building both ways simultaneously with a P node

matched symbol. Not shown in the figure is the alternative that the model position is deleted. In this case, a single transition is made from the T state to the D state. The mapped sequence length remains unchanged and the score increases by the T-to-D transition score only.

The effect of moving from a top-level state T in a child node to the top-level ML state in a parent L node is shown in Figure 7.8 and is an exact mirror of the situation in Figure 7.7 for an R parent node. It can be noted that structure generated by a model with only a single branch of R nodes or L nodes is very similar to the profile HMM. Each node contains one match, one insert, and one delete state. In fact, the single-emission nodes of the CM are simply modeling the primary sequence homology of the non-base-paired portions of consensus alignment. Also note, there is no need to have IR states in L nodes or to have IL states in R nodes. Insertions to the outside of the sequence represented by a top-level state are generated by the next node up the tree (possibly by the root start node, which is why the root start node has both IR and IL states).

All of the advantage of using a CM over an HMM is embodied in the MP states of the P nodes. It is the sixteen distinct emission probabilities for each of the possible nucleotide pairs that allows covariation to be detected. Typically, there will be one pair with very high probability (indicating both sequence homology and secondary structure homology) and other canonical or wobble base pairs with lesser, but still high, probabilities. Figure 7.9 shows the effect of a transition from a child-node top-level state T to the MP state of a parent P node. The IL state can be visited zero or more times and the IR state can be visited zero or more times in between. The number of IL visits does not need to equal the number of IR state visits. Even though MP appears twice in Figure 7.9, this is a single visit of the MP state which emits two match symbols. The score increases by the sum of the transition score and the emission score for each of these transitions.

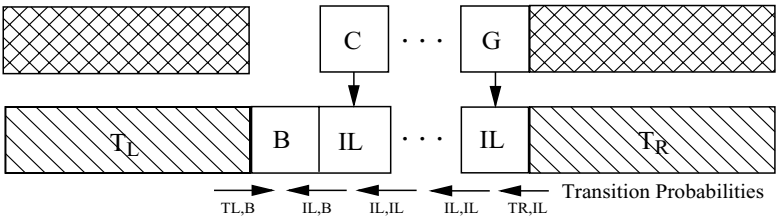

Fig. 7.10. Joining two submodels with a B node

Using the S, L, R, P, and E nodes it is possible to construct a single branch of a CM tree that can describe any secondary structure as long as all base pairs are fully nested. For instance, the structures «–«–––»>–> and ««<––»–> can be described, but «–»–««––»» and «–»««<––» >–«–» can not. The later two structures require one and two bifurcations respectively. Roughly speaking, the branches of the CM model tree correspond to stem-loops in the RNA secondary structure. Figure 7.10 shows how a B state joins two submodels represented by its left and right child states (the TL and TR states). These states are always start states and are part of a left-child S node and a right-child S node respectively. It is necessary that either the left-child S node contain an IR state or the right-child S node contain an IL state in order to allow inserted database symbols to the right of the rightmost consensus position of the left-child submodel. Since left emissions are always preferred to right emissions by tradition whenever either would be possible, right-child S nodes are chosen to contain an IL state and left-child S nodes do not contain any insert states.

Depending on the number of insert and delete state visits the length of the sequence mapped by a transit of the model can be longer or shorter than that of the consensus sequence. In order to find the optimal mapping of a database sequence to the model, one should consider all possible insertion and deletion patterns. If the database sequence is an entire chromosome, the database sequence could be several hundred million bases in length. It would be possible in theory to search insertion patterns that include total inserted symbol counts of many millions of database symbols. In practice, true RNA genes are unlikely to have so many insertions and even if they did, they would be rejected by a scoring scheme that would add up large numbers of gap extension penalties. In order to make database search possible with deterministic algorithms such as dynamic programming, dynamic programming a cutoff on the maximum database sequence length that can be represented at any state in the model is made. Often this cutoff is less than one and one half times the consensus sequence length of the RNA family. The length of the database sequence mapped to a CM state is equal to the consensus sequence length represented by the model subtree rooted at that state plus the net number of insertions less deletions made up to that state.

The traditional method of using a CM for database search is to use dynamic programming. A score is calculated for the database subsequences ending at every possible position in the database and for every possible subsequence length up to the length cutoff discussed above. At each possible database position, the maximum over all subsequence lengths explored for that position is taken as the database position

score. Database position scores exceeding a selected threshold cause the database position to be declared as the ending position of a putative gene for the RNA family represented by the CM.

The dynamic programming algorithm starts at the E states. These states represent null sequences and are assigned a score of zero for all possible database subsequence end positions and the mapped subsequence length is taken to be zero. Score evaluation then progresses from the E states up the tree towards the root S state. The scores at the root S state are the scores used to determine the putative gene locations. At each model state, the best possible score for a database subsequence ending at each possible database position and for every possible database subsequence length between zero and the length cutoff are evaluated. These best scores are used by parent states to calculate the best possible scores for the submodel rooted at the parent node.

A more formal description of the dynamic programming algorithm used to generate scores for a database sequence with respect to a covariance model of an RNA family is given in Figure 7.11. The algorithm uses a triple-nested loop over database end position, subsequence length, and CM state. All database end positions **j** in the range 0 to the length of the database sequence **L** are examined. All database subsequence lengths in the range 0 to the length cutoff **D** are examined (with subsequence lengths that would extend past the start of the database sequence ignored). Finally scores are generated for each state number **v**, where there are $\mathbf{M}+\mathbf{1}$ states and the root start state is numbered 0. States are indexed such that the index of a child state is always higher than that of a parent state. Evaluating state scores in reverse state index order ensures that the score for a submodel at a child state is always available when needed by its parent state. The scores calculated at each position, subsequence length, and state are given by $\mathbf{s}(\mathbf{j},\mathbf{d},\mathbf{v})$.

For E states, the score is zero for null subsequences and minus infinity for any other subsequence. This ensures that any subsequence other than the null subsequence is discarded by the maximum operation when finding the best score further up the tree. Delete and start states are computationally identical. Neither adds any database symbols to the mapping. Only the transition score $\mathbf{trans}(\mathbf{c},\mathbf{v})$ from the child

```
for j = 0 to L
 for d = 0 to min(D,j)
  for v = M to 0
   case type(v) is
    E: if d == 0 then s(j,0,v) = 0; else s(j,d,v) = -Infinity
    D or S: max(over children c)[s(j,d,c) + trans(c,v)]
    L: if d == 0 then s(j,0,v) = -Infinity; else s(j,d,v) =
     emit(l,v) + max(over children c)[s(j,d-1,c)+trans(c,v)]
    R: if d == 0 then s(j,0,v) = -Infinity; else s(j,d,v) =
     emit(r,v) + max(over children c)[s(j-1,d-1,c)+trans(c,v)]
    P: if d < 2 then s(j,d,v) = -Infinity; else s(j,d,v) =
     emit(l,r,v) + max(over children c)[s(j-1,d-2,c)+trans(c,v)]
    B: max(over k in 0 to d)[s(j-k,d-k,lc)+s(j,k,rc)]
```

Fig. 7.11. Algorithm for dynamic-programming CM scoring

state **c** to parent state **v** is needed here. The L, R, and P state types add database symbols to the mapping and therefore change the database position and/or subsequence length mapped when compared to the mapping passed from the child state. L-type states include both IL and ML states, R-type states include both IR and MR states, and P-type states only appear as MP states. L- and R-type states add one symbol to the mapping, so the resulting mapping can not have length less than 1. Therefore the score is set to minus infinity for length 0. P-type states add two symbols to the mapping and therefore a score of minus infinity (meaning impossible) is set for lengths of 0 or 1. For L-type states, the resulting database end position remains unchanged, but mapping a symbol on the left increases the subsequence length by one, so the score for length **d** depends on a child score for length $\mathbf{d}-\mathbf{1}$. For R-type states, adding a symbol on the right moves the end position one place right, so the score for length **d** and position **j** depends on child length $\mathbf{d}-\mathbf{1}$ and position $\mathbf{j}-\mathbf{1}$. For P-type states, the score for length **d** and position **j** depends on child length $\mathbf{d}-\mathbf{2}$ and position $\mathbf{j}-\mathbf{1}$. For L-, R-, and P-type states, the change in score is both an emission score (**emit**) and a transition score (**trans**). The emission score depends on the database symbols found on the right (**r**) and left (**l**) respectively. Finally, bifurcation finds the best two submodels whose lengths add up to the subsequence length score to be evaluated and which are contiguous along the database.

It should be clear that the amount of computation involved in the dynamic programming algorithm is very large. In fact, this is the central drawback to application of the algorithm. The remainder of this chapter explores current use of covariance models for RNA gene finding and approaches to reducing the computational cost of CM-based database search.

7.3 Application of CM-Based RNA Gene Finding: The Rfam Database

Over five hundred families of non-coding RNA sequences have been identified and modeled in the publically-available Rfam database [16]. Groups of sequences are formed into families using reference to the literature, aligned, and annotated with secondary structure using either experimentally-derived or computer-predicted structures. These carefully hand curated multiple alignments are referred to as "seed" alignments in Rfam. The structure of the CM model tree is then generated from the consensus secondary structure annotation of the alignment. Transition and emission scores for the CM are estimated from observed frequencies of nucleotides and gaps at the various multiple alignment columns. A prior distribution is then combined with the observed frequencies in an attempt to correct for limited sample size. This has the effect of eliminating log-likelihood ratio scores that are minus infinity due to observed counts of zero. Such scores are undesirable because they rule out certain patterns merely because they did not happen to be observed in the training data even though there is no theoretical reason to believe that they are impossible.

A package of programs used by Rfam to estimate covariance models and for database search is called Infernal [17]. This package is publicly available, including

source code. The Rfam site includes both alignments of the original seed sequences for families and combinations of seed sequences and new sequences found through database search. The database also includes the parameter files of the estimated covariance models. These parameter files are particularly useful when exploring methods other than dynamic programming for CM-based database search since the parameters can be transformed into other formats suitable for alternative searches.

The amount of computational power needed for direct search of the available genomic data using dynamic programming and covariance models is excessive. In order the trim the amount of data searched by orders of magnitude, a filtering operation is first applied to the database. In the case of Rfam, the filtering method is to use BLAST [8] to score the database with respect to the consensus sequence of the model. It is hoped that the new RNA genes will have enough primary sequence homology with the existing family members that their score will be raised enough above the background noise to be retained in the portion of data passed to the full CM search. The extent to which this hope is true in practice is not very well studied. In the following section of this chapter, another proposed filtering method from the literature will be discussed. In section 4, we will discuss an evolutionary computation alternative to filtering the database (and to using preset length cutoffs).

In examining the possibility of using evolutionary computation, data extracted from the Rfam database is used. In particular, fourteen sequences belonging to the U12 ncRNA family (accession number RF00007) are used for testing. The parameter file for this Rfam family is also used. Of the fourteen sequences, seven are from the seed family used to estimate the model parameters and seven were found through database search using the model. The U12 family [18] [19] [20] are small nuclear RNA (snRNA) which form a complex with specific proteins to function as part of the minor spliceosome. The function of U12 is to remove introns from pre-mRNA. The U12 ncRNA acts in a way similar to that of the U2 ncRNA in the major spliceosome.

7.4 Filters to Reduce Database Search Cost

A major problem with using a filter to reduce the amount of genomic data to be searched with a CM is that there may not be enough primary sequence homology to keep the true gene in the retained data set. With the BLAST method of database reduction, there is no known way to set the score threshold to guarantee retention. However, Weinberg and Ruzzo [21] have recently come up with a way to guarantee that a profile HMM filter will not discard any portion of the database that contains a subsequence that the dynamic- programming CM search would score as a putative RNA gene. The procedure involves extracting from the CM parameter files equivalent profile HMM parameters that ignore the joint probability information inherent in the P state emission probabilities. The maximum additional score that could come from the secondary structure information in the CM with a perfect database match can be calculated and subtracted from the score threshold to be used with the CM search. The result is the minimum primary sequence score contribution that must come from the database sequence in order for the overall CM score to exceed the

CM threshold. Portions of the database which do not meet this minimum score contribution are found when the HMM score does not exceed this minimum primary sequence contribution.

Disadvantages of the HMM method are that the HMM is much slower than BLAST (although significantly faster than full CM search) and that the reduction in database size varies greatly from one RNA family model to another. No comprehensive study of the speedup of this method has been undertaken. The Weinberg and Ruzzo paper looks at only 34 of the over 500 families. Extrapolating from this paper, it is still predicted to take tens of CPU years with a modern desktop computer to search all Rfam families on the 8-gigabase database of the study. Since both the number of known RNA families and the amount of genomic data are rapidly expanding, this amount of computation is still too much.

Weinberg and Ruzzo [22] have also recently come up with an heuristic filter that is an alternative to BLAST and appears to perform better than BLAST. This heuristic filter is based on a profile HMM and as such does not use secondary structure information at the filtering stage.

7.5 An Alternative to Filters Using Evolutionary Computation

In this section, the use of evolutionary computation (EC) as an alternative to filtering followed by dynamic programming search is examined. Secondary structure information will be used from the start on the entire database. Also, no sequence length cutoff is employed. The results of the non-exhaustive EC-based search will also be compared to those of a simple hill-climbing algorithm which is also non-exhaustive, but does not have the ability to escape local minima. The ability to jump out of a local minimum is shown to be crucial to the algorithm.

7.5.1 Dynamic Programming Efficiency

To motivate why the traditional dynamic-programming exhaustive search might not be the most efficient way to find RNA genes using a CM, the observed usage of search space regions is first examined. Dynamic programming finds the best score at each model state for each database end position and each database subsequence length ending at that position (up to a predefined cutoff length). The first observation made is that only a small range of the subsequence lengths evaluated at a given state are normally observed in real data. These subsequence lengths cluster about the consensus sequence length for the submodel represented by the subtree rooted at the state. In what follows, *length deviation* will be defined as the length of the database subsequence generating a score at a given state minus the length of the consensus sequence represented by the state. Length deviation is therefore equivalent to the number of inserted symbols minus the number of deleted model positions in the submodel mapping of a given state.

The actual usage of subsequence lengths at various states of the Rfam U12 CM model for the fourteen known U12 family members (seven seed and seven discovered members) is shown in Table 7.1. The “top” and “bottom” designations and the

Table 7.1. Subsequence length use in observed U12 data

State	Branch	Consensus Length	Obs Max	Obs Min	DP Max	DP Min
root S	a	149	+6	−6	+11	−149
bottom R	a	145	+6	−6	+16	−145
top P	b	20	+4	−3	+140	−20
bottom L	b	1	+1	0	+159	−1
top L	c	47	+6	−4	+113	−47
bottom L	c	40	+6	−4	+120	−40
top P	d	22	+6	−1	+138	−22
bottom L	d	1	0	0	+159	−1
top P	e	17	0	−3	+143	−17
bottom L	e	1	0	0	+159	−1
top L	f	77	+1	−3	+83	−77
bottom L	f	67	+1	−2	+93	−67
top P	g	37	+1	−1	+123	−37
bottom L	g	1	0	0	+159	−1
top L	h	29	0	−2	+131	−29
bottom L	h	1	0	0	+159	−1

branch letter refer to Figure 7.3. For example, "bottom R" and branch "a" is the consensus MR state in the R node and the bottom of the "a" branch of the CM tree. The consensus sequence length is 145 for the model subtree rooted at this MR state since two more R nodes and two more L nodes are in the CM tree above it and the overall consensus sequence length is 149. Using dynamic programming, all subsequence lengths in the range 0 to 160 are investigated at every state (since the cutoff length for this model is chosen to be 160 in Rfam). The last four columns of the table show length deviations from the consensus length at each model state. The dynamic programming (DP) maximum and minimum length deviations are always 160 minus the consensus length and the negative of the consensus length respectively. The observed maximum and minimum length deviations are shown in the fourth and fifth table columns respectively. These are seen to cluster near zero and to be generally much smaller than the dynamic programming limits.

One possible way to make the dynamic programming algorithm more efficient by about one to two orders of magnitude is to specify state-dependent minimum and maximum length deviations (or equivalently, minimum and maximum subsequence lengths). This requires extra complexity in the search code. The model input file needs to be augmented with the state-dependent search limits. These limits would need to be determined by a program that automatically extracts the observed length deviations at each state from the seed sequence multiple alignment. A buffer region about the observed deviations needs to be added to allow for deviations of true family members that are outside the range observed in the seeds. The statistical analysis needed for a good choice of buffer region size is nontrivial.

7.5.2 CM-Based Search Without Dynamic Programming

Another way to improve efficiency is to expand the search about the zero length deviation solution [23]. This could be done with either deterministic (such as hill-climbing) or randomized (such as genetic algorithm) search methods. In either case, the initial step is to determine the scores of an ungapped mapping of a database subsequence to the covariance model at every database position. This is equivalent to evaluating every consensus state for zero length deviation only and assigning a score of minus infinity if the length deviation is not zero or the state is a non-consensus state. Unlike filtering with BLAST, Fasta, or an HMM, the ungapped scoring method employs base-pairing information from the start on all portions of the database. It is also several orders of magnitude faster than the full dynamic programming CM search due to a number of factors. First, only consensus states are evaluated for about a factor of three reduction in evaluated states. Second, there is no need to add state transition scores in this initial sweep of the database since they only contribute an additive constant to the score at every database position, for a computational reduction of about a factor of two. Third, only one subsequence length is evaluated for a reduction by a factor of the cutoff length (often two to three orders of magnitude). Forth, bifurcation states are very expensive relative to other states since they need to check every possible allotment of subsequence length between the two branches. This results in bifurcations having a computational complexity that is higher than other states by a factor about equal to the cutoff length. Since the function of the bifurcation is not needed without gaps, this saves another two to three orders of magnitude. Overall, ungapped scoring of a database is somewhere in the range of three to eight orders of magnitude faster than full dynamic programming scoring. The proportionate speedup is greater for models with very long consensus sequences (and long length cutoff), which are exactly the models that take longest with the conventional scoring method.

The clustering of the true subsequence lengths about zero length deviation as shown in Table 7.1 for U12 is a general phenomenon. What is less clear is whether the scores of the best solutions improve monotonically as the length deviation is changed from zero to its true value. If all possible insertions of two contiguous symbols are attempted and all possible deletions of three contiguous symbols are attempted, then a large number of scores are generated for length deviations of $+2$ and -3. If all possible combinations of simultaneous double insertions and triple deletions are tried, then a much larger number of scores for length deviations of -1 are created. The alignment patterns with the double insertion and triple deletion may be much different than that of a single deletion (which also has a length deviation of -1). Thus it is not clear that an algorithm that searches by trying every possible single insertion and every possible single deletion at each model position will necessarily move closer to the true solution. In fact, it has been found for the U12 family that such a simplistic hill-climbing approach does not work as well as a randomizing algorithm that is capable of escaping from local minima.

In addition to the possibility of focusing the search on model mappings with relatively few insertions and deletions, there is also the possibility of focusing the

search around database locations that have high scores with suboptimal alignments. The initial ungapped sweep of the database should give generally higher scores near true RNA gene family members than on unrelated portions of the database. This is a result of matching at least some part of the database sequence that does not happen to have gaps relative to the consensus sequence. The search can start with an expansion about the ungapped alignment for a relatively large number of high-scoring database positions. As some database positions start to show score improvements and others not, the search can move to focus only on those database positions showing either very high initial scores or somewhat lower, but improving, scores. Finally, once the number of database positions in narrowed sufficiently, it is possible to resort to full dynamic programming search of the neighborhoods around the very highest scoring positions.

It is helpful to have a fixed-length representation of the alignment of a database subsequence to the consensus sequence of the CM. The representation used here is taken from the literature on protein threading using evolutionary computation [24]. A vector of non-negative integers of length equal to the length of the consensus sequence is used. If a vector element is 0, then the corresponding consensus model symbol is deleted. If the vector element is 1, then the model symbol is matched and there are no inserted database symbols to the right of this consensus position. If the vector element is a value $\mathbf{n}$ greater than 1, then $\mathbf{n}-\mathbf{1}$ database symbols are inserted. Figure 7.12 shows the correct alignment vectors for the seven seed sequences of the U12 family (see Figure 7.1 for comparison). Each alignment is a vector of 149 integers and the break of six spaces is only in the figure to show correspondence between the values and the original multiple alignment (the actual representation contains no such spaces). The goal of the search algorithm is to expand the search around the initial solution vector (149 ones) toward the true alignment vectors as in Figure 7.12. Notice that there is nothing more to do in the case of database sequences 2 and 6 since the optimal alignment has no gaps with respect to the consensus sequence. There is no way to represent insertions to the left of the first position, so the representation is the same whether the sequence has a symbol in the first column or not. This is not a problem, since the alignment is local. The putative gene start position can be off by several bases due to initial insertions.

There are two components to a candidate search solution, the alignment vector and the location in the database sequence of the first alignment position. An alignment vector change is assumed to take a form that results in either adding or removing one or more contiguous insertions or deletions. Adding contiguous insertions involves increasing a single vector element by one or more. Removing contiguous insertions is done by decreasing a single vector element by one for more such that the resulting element value is greater than 0. Adding contiguous deletions is accomplished by changing a range of consecutive vector elements to 0. Removing contiguous deletions occurs when a range of consecutive zeros are all changed to 1.

When considering a change to a given candidate solution to potentially improve the score of the solution, two classes of variations are possible which will be called compensating and non-compensating. Non-compensating changes are those for which the alignment of the model to the database remains unchanged to the left

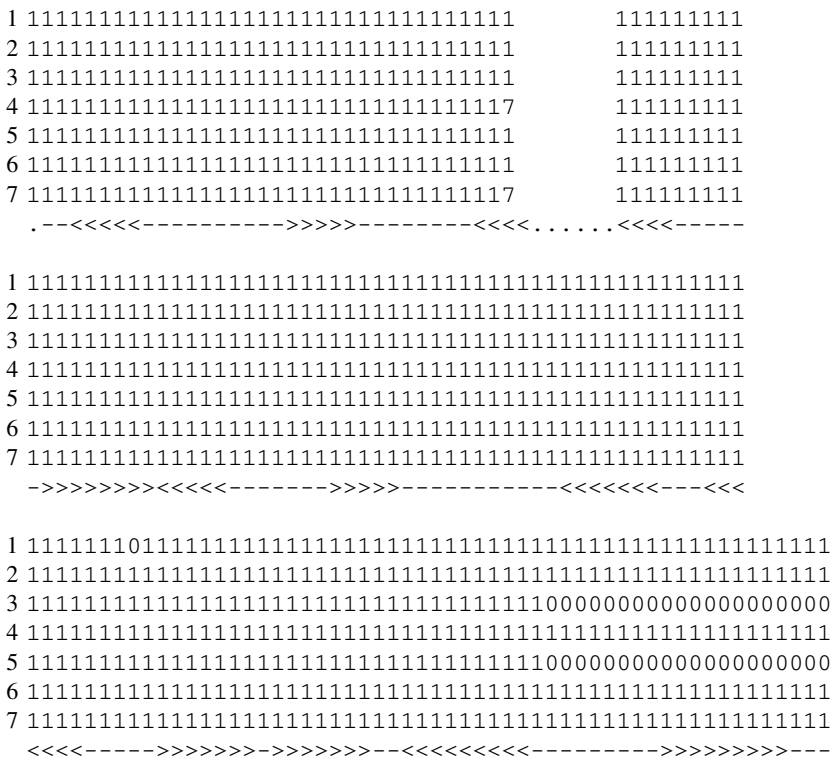

Fig. 7.12. U12 seed sequence alignment vectors

of the alignment vector alteration and compensating changes are those for which the model/database alignment is unchanged to the right. Non-compensating changes are the result of alignment vector changes with the database location of the first position unchanged. Compensating changes occur when the database start position is changed by an amount with equal magnitude but opposite sign to that of the total change in alignment vector values. Figure 7.13 shows how a compensating change to a candidate solution might improve a score whereas a non-compensating change does not. Initially the alignment vector is **11111111** and the database start position is at the first A in the portion of the database sequence shown (**GGAAUCACUG**) as shown at the top of the figure. The correct alignment vector is **11131111** and the correct database start position is two places further to the left. The initial alignment causes the last four database symbols shown to correctly align with the last four consensus symbols of the model. If a close (but not exactly correct) change to the alignment vector is tried such that the candidate vector is **11311111** without a compensating change to the database start position, the alignment gets worse. This non-compensating change is shown in the middle of the figure. If the same close change to the vector is tried, but with database start position compensation, then the situation is as in the bottom portion of the figure. Since the sum of the vector element values increased by 2

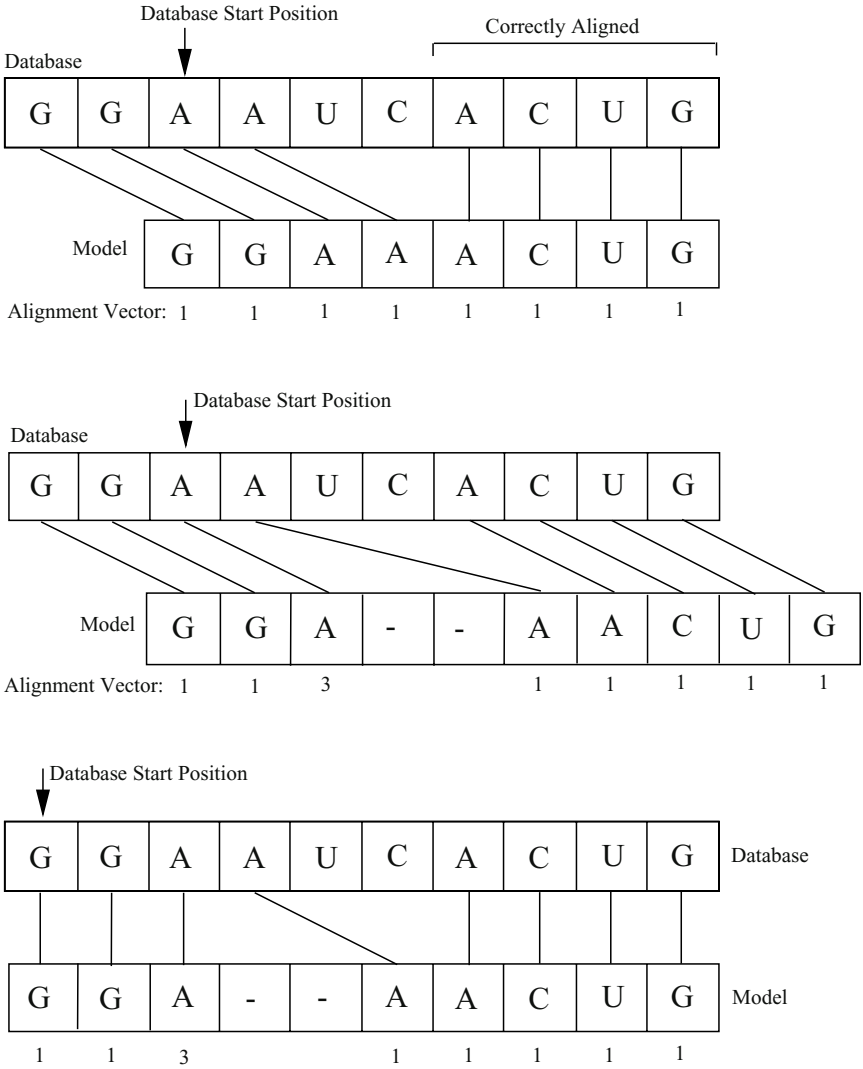

Fig. 7.13. Compensating and non-compensating candidate solution changes

when going from **11111111** to **11311111**, the compensating change is to decrease the starting position by 2. Now all but one of the positions aligns and the score should increase. Sometimes, the portion of the alignment that is contributing to a high score for the initial solution is due to a good alignment to the left of the vector change and uncompensated change may improve a score. Other times, the high-scoring alignment portion may be to the right of the vector change and compensated change may improve the score. In general, both should be tried.

The fitness function of an individual is the score of the database subsequence associated with the individual with respect to the covariance model. The database subsequence associated with an individual starts at a database location specified by the individual and continues for a length equal to the sum of element values of the alignment vector. The alignment vector specifies a unique path through the covariance model states and the score is found as the sum of log-likelihood ratio scores for each of the state transitions and symbol emissions for this unique tree parse. The search space is the set of all possible starting locations within the database as well as all possible combinations of non-negative alignment vector values such that the sum of the vector values plus the starting location does not exceed the end of the database. Single-point crossover is employed and as well as single mutations. The single mutations take either the form of changing a single alignment vector element to some other non-negative value (changing the number of insertions at a point) or taking a range of values and changing them to 0 (creating a contiguous deletion region). Single mutations can either be compensating or non-compensating as described above with a fifty percent probability of each likely a good choice. The probabilities of single mutations to small element values should be higher than those of large values (an exponentially decreasing probability would be a reasonable choice). Similarly, the probabilities of small deletion regions should be greater than large regions.

7.5.3 Experimental Results

In order to try out the idea of using a GA to search for good CM alignments in a database, an artificial dataset has been created which contains a mixture of U12 RNA genes and other ncRNA genes. This was done to keep the test database small and at the same time provide tempting incorrect targets for the algorithm searching for U12 genes. The other ncRNA genes contain stem-loop structures that are likely to be more similar to U12 genes than randomly chosen segments of genome. In future research, the GA search method should be applied to a real search, but this research has not yet progressed to that point. This is partly due to the fact that the current version of the software is written in MATLAB and needs to be rewritten in C and optimized for large-scale use.

The test dataset contains 15880 bases, such that about ten percent of the sequence is composed of true U12 genes and the remainder of randomly selected other ncRNA genes taken from Rfam. Since the true U12 genes are in the Rfam database, they are all able to pass the BLAST filter (true genes that might exist and can not get past the filter can not be in Rfam by definition). The initial ungapped scoring of each database position with the CM for U12 is shown in Figure 7.14. The true U12 genes start at database locations 446, 1039, 2475, 3858, 6096, 7406, 8196, 8880, 9705, 10774, 11624, 12428, 13493, and 14615. Twelve of these locations have peaks in the initial database sweep at or near the true position with scores that are the twelve highest scores out of the 15732 scores (148 positions at the end of the database are not scored with respect to the 149 position model). Two of the true U12 genes have peaks that are harder to discern from the background. The peak at 8196 has a score of about 5

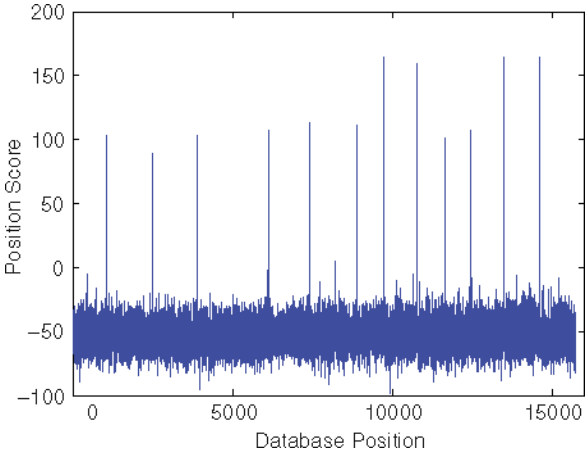

Fig. 7.14. Ungapped scores at each database position

and is the 17th highest peak and the peak at 446 has a score of just under 0 and is the 22nd highest peak. These scores are base-2 log values and therefore have units of bits. Neither a score of 0 nor 5 would be considered statistically significant in any reasonable database search. A search algorithm that can generate a better alignment for these two marginal cases is of primary interest, although improvements in the alignment of the other twelve is also indicative of a generally successful algorithm. The U12 gene at 446 was a seed sequence used to estimate the CM, but the gene at 8196 was not. Four of the true U12 genes do not have any gaps with respect to the consensus sequence and therefore the score from the initial ungapped database sweep is already optimal.

The 100 highest-scoring locations from the upgapped database scores are used as starting points for two search algorithms. The first is a simple hill-climbing algorithm and the other is a genetic algorithm. Each algorithm is permitted 700 candidate solution evaluations per starting point. The GA is run for 20 generations with 35 individuals per generation and the hill-climbing algorithm is run for six rounds of 116 evaluations each. It turns out that additional rounds for the hill-climbing algorithm would not be helpful since the algorithm has converged on a solution in all 14 true U12 gene cases by the sixth round. The choice of 700 evaluations per location is based on observed convergence of the GA.

With a step size of 1 for alignment vector changes in the hill-climbing algorithm, there are 4*149 = 596 possible changes per round. These changes are to increase or decrease a single vector element by 1 without start location compensation and increase or decrease a single element with compensation. Since this would use up almost all of the allocated evaluations in a single round, an alternative strategy is employed. Every fifth alignment vector element is allowed to change rather than every element. This can result in slightly suboptimal alignments if a true insertion or deletion is not at the allowed change location, but the suboptimality is small (see

Figure 7.13 for the effect of making a change in a position slightly different than the correct position). Using every fifth position results in 4*29 = 116 evaluations per round.

The GA uses both mutation and single-point crossover to create new individuals. The fittest individual is retained in the new generation (elitism). Four new individuals per generation are produced by single-point crossover without mutation of two individuals randomly selected from the twenty fittest with crossover points uniformly distributed along the alignment vector. All remaining individuals are produced using mutation without crossover. Twenty five individuals per generation are generated with a single mutation of an individual chosen randomly from the five fittest, where the mutation takes the form of increasing or decreasing an alignment vector position by 1. Half of these mutations are randomly selected to be compensating and the other half non-compensating. Finally, five individuals are produced each generation with a single mutation uniformly chosen in the range $+7$ to -1. Half of these mutations are compensating and half not.

Table 7.2 shows the scores of the best solutions in the final round or generation for the hill-climbing and GA algorithms. The hill-climbing algorithm is deterministic, so only one run is made because the result is always the same. The GA result is the mean over ten runs of the best solution in the final generation. The table also shows the source of the U12 gene sequence in terms of EMBL accession code and the nucleotide positions of the gene within the EMBL sequence. The seven seed sequences are identified with a cross-reference number to the sequences shown in Figure 7.2 of this chapter.

Overall, the performance of the two algorithms is rather similar. However, the two genes of greatest interest 442 and 8196 show that the hill-climbing algorithm did not improve the scores at all, whereas the GA made significant gains. The scores

Table 7.2. Experimental results on U12 dataset

Dataset Position	Hillclimb Score	GA Score	Accession Code (EMBL)	Nucleotide Positions	Fig 7.2 Seed No.
446	−0.74	43.68	L43844.1	2–149	1
1039	146.37	145.66	AC087420.4	142608–142466	
2475	124.95	123.88	AC112938.11	234142–234291	
3858	146.37	143.34	AL591952.9	131760–131611	
6096	110.92	133.57	AL669944.8	2483–2625	
7406	159.12	158.12	AC133939.4	22042–22191	
8196	5.24	40.85	AC132590.3	81080–80927	
8880	147.13	147.13	AL772347.6	146375–146226	
9705	164.47	164.47	L43843.1	2–150	2
10774	159.12	159.12	L43846.1	332–480	3
11624	160.80	160.30	J04119.1	2–150	7
12428	110.92	125.88	L43845.1	358–512	4
13493	164.47	164.47	Z93241.11	76642–7679	6
14615	164.47	164.47	AL513366.11	57717–57871	5

of the optimal alignments for these two sequences are 78.48 and 81.79 respectively, so the GA was only able to get about half of the score increase possible when measured in bits. Even so, the scores changed from statistically insignificant to very statistically significant. Two other cases, sequences 6096 and 12428 also show more improvement with the GA than with the hill-climbing algorithm. These results seem to indicate that there is some advantage to an algorithm with randomization that can jump out of local optima when doing CM-based RNA gene search.

7.6 Conclusions and Future Direction

We have seen that traditional dynamic programming scoring of database sequences with respect to ncRNA gene family covariance models can be rather inefficient due to consideration of may candidate alignment solutions that are far different that those observed in real genomic data. Dynamic programming scoring also requires the use of an arbitrary cutoff on the maximum allowed length of putative genes in the database and a primary-sequence-only filtering of the database in order to reduce required computational resources to a feasible level. Both the cutoff and filtering can cause loss of sensitivity. Dynamic programming further applies equal computational effort to all portions of the database retained by the initial filtering operation.

An alternative scoring method using genetic algorithms does not impose a length cutoff and uses the secondary structure information in the covariance model parameters right from the start. This method also has the potential to allow regions of the database which are not showing score improvement to be abandoned before excessive computational resources are applied in those regions. An exploratory investigation of the alternative scoring method has been applied to a set of known U12 genes and the results are encouraging. This experiment also gives some evidence that deterministic search algorithms which can not escape local optima may not be successful.

Much remains to be done to turn this alternative scoring idea into a standard functional RNA gene search methodology. Investigations on more ncRNA gene families need to be undertaken to determine how to best choose which database locations should be passed to the search algorithm based on the initial ungapped scan of the database. The details of when to abandon a search at a given database location need to be worked out. The experimental investigation above did not even attempt this as a fixed number (700) of evaluations was undertaken at each position. Other stochastic search methods such as simulated annealing need to be investigated to see if they might outperform the genetic algorithm. A parameter sweep needs to be undertaken for such things as the optimal number of individuals per generation and the ratio of crossed-over to mutated individuals.

Improvement to the search may also take the form of better direction for the mutation operator. There is information in the covariance model parameters as to the relative likelihood of an insertion or deletion at a particular point in the consensus sequence of the model. This information could be used to make mutations at these locations statistically more likely during mutation. Also, it may be possible to guess

good mutation points by examining the contribution to the overall score of a candidate solution as a function of location in the consensus sequence normalized to the maximum score possible at the location. A drop off in this score-contribution measure at a particular location may be indicative of an insertion or deletion at that location.

In general the use of covariance models for RNA gene search is not nearly as well developed as the use of profile hidden Markov models for protein domain classification. With the increasing recognition of the importance of untranslated RNA to biological function, there should be significant interest in computational methods to study the function and structure of these molecules.

References

1. International Human Genome Sequencing Consortium (2004) Finishing the euchromatic sequence of the human genome. Nature 431:931–945
2. Gesteland R, Cech T, Atkins J (2006) The RNA world. Cold Spring Harbor Laboratory Press, New York
3. Burge C, Karlin S (1997) Prediction of complete gene structures in human genomic DNA. J Mol Biol 268:78–94
4. Eddy S (1998) Profile hidden Markov models. Bioinformatics 14:755–763
5. Finn R, Mistry J, Schuster-Bŋockler B, Griffiths-Jones S, Hollich V, Lassmann T, Moxon S, Marshall M, Khanna A, Durbin R, Eddy S, Sonnhammer E, Bateman A (2006) Pfam: clans, web tools and services. Nucleic Acids Research 64:D247–D251
6. Rivas E, Eddy S (2000) Secondary structure alone is generally not statistically significant for detection of noncoding RNAs. Bioinformatics 16:583–605
7. Durbin R, Eddy S, Krogh A, Mitchison G (1998) Biological sequence analysis. Cambridge University Press, Cambridge UK
8. Altschul S, Gish W, Miller W, Myers E, Lipman D (1990) Basic local alignment search tool. J Mol Biol 215:403–410
9. Pearson W, Lipman D (1988) Improved tools for biological sequence comparison. Proc Natl Acad Sci 85:2444–2448
10. Smith T, Waterman M (1981) Identification of common molecular subsequences. J Mol Biol 147:195–197
11. Chomsky N (1959) On certain formal properties of grammars. Information and Control 2:137–167
12. Eddy S, Durbin R (1994) RNA sequence analysis using covariance models. Nucleic Acids Research 22:2079–2088
13. Zucker M (1989) Computer prediction of RNA structure. Methods in Enzymology 180:262–288
14. Wiese K, Hendricks A, Deschênes A, Youssef B (2005) Significance of randomness in P-RnaPredict - a parallel algorithm for RNA folding. IEEE Congress on Evolutionary Computation
15. Eddy S (2003) HMMER user's guide. http://hmmer.janelia.org
16. Griffiths-Jones S, Moxon S, Marshall M, Khanna A, Eddy S, Bateman A (2005) Rfam: annotating non-coding RNAs in complete genomes. Nucleic Acids Research 33:D121–D124

17. Eddy S (2005) Infernal user's guide. ftp://selab.janelia.org/pub/software/infernal/Userguide.pdf
18. Shukla G, Padgett R (1999) Conservation of functional features of U6atac and U12 snRNAs between vertebrates and higher plants. RNA 5:525–538
19. Tarn W, Steitz J (1997) Pre-mRNA splicing: the discovery of a new spliceosome doubles the challenge. Trends Biochem Sci 22:132–137
20. Otake L, Scamborova P, Hashimoto C, Steitz J (2002) The divergent U12-type spliceosome is required for pre-mRNA splicing and is essential for development in Drosophila. Mol Cell 9:439–446
21. Weinberg Z, Ruzzo W (2004) Faster genome annotation of non-coding RNA families without loss of accuracy. Int Conf Res Computational Molecular Biology (RECOMB) 243–251
22. Weinberg Z, Ruzzo W (2006) Sequence-based heuristics for faster annotation of non-coding RNA families. Bioinformatics 22:35–39
23. Smith S (2006) Covariance searches for ncRNA gene finding. IEEE Sym Computational Intelligence in Bioinformatics and Computational Biology (CIBCB) 320–326
24. Yadgari J, Amir A, Unger R (2001) Genetic threading. Constraints 6:271–292