2022

FATHMM: Frameshift Aware Translated Hidden Markov Models

Genevieve Krause
University of Montana, Missoula

FATHMM: FRAMESHIFT AWARE TRANSLATED HIDDEN MARKOV MODELS

By

Genevieve R. Krause

Bachelor of Arts, Prescott College, Prescott, AZ, 2006

Thesis

presented in partial fulfillment of the requirements
for the degree of

Master of Science
in Computer Science

The University of Montana
Missoula, MT

Spring 2021

Approved by:

Ashby Kinch Ph.D., Dean
Graduate School

Travis Wheeler Ph.D., Chair
Computer Science

Doug Brinkerhoff Ph.D.
Computer Science

Mark Grimes Ph.D.
Biological Sciences

Krause, Genevieve R., M.S., May 2021 Computer Science

FATHMM: Frameshift Aware Translated hidden Markov Models

Chairperson: Travis Wheeler

Accurate annotation of biological sequences is fundamental to modern molecular biology. Tools such as BLAST and HMMER which can quickly and accurately annotate many types of biological sequences by aligning them to known sequences or sequence models. The work described in this thesis is motivated by the goal of annotating protein-coding DNA, particularly for genes that are highly diverged from known sequences. I demonstrate that gains in annotation sensitivity are achieved both by aligning protein-coding DNA directly to protein sequences (so called "translated alignment") and through the use of profile hidden Markov models (pHMMs). These pHMMs provide position-specific patterns of sequence conservation and enable computation of annotation support by summing over all possible sequence alignments.

Even with pHMMs, translated annotation of protein-coding DNA sequences containing frameshift inducing indels can be particularly troublesome, as standard models do not support alignment through frameshifts. Here I present FATHMM: Frameshift Aware Translated Hidden Markov Models. This translated sequence alignment tool produces high-quality translated alignments and accurate annotation for even heavily frameshifted DNA sequences. Built within a fork of the open source HMMER software package, FATHMM is based on a new frameshift-aware pHMM architecture, and includes a complete frameshift-aware re-implementation of HMMER's Forward and Backward algorithms, posterior probability calculations and alignment recovery, and score adjustment for composition bias. FATHMM promises to increase annotation of sequences that have endured frameshifts through neutrally-selected mutation (such as pseudogenes and transposable element proteins), as well as sequences containing frameshifts due to sequencing error.

ACKNOWLEDGMENTS

First I would like to acknowledge my son, Oscar, who is my inspiration in all things. He taught me the meaning of perseverance and I strive to be someone he can be proud to call his mom.

I would also like to thank all the members of the Wheeler lab, past and present, who have worked along side me in the completion of this work. I have been blessed to have such ready access to so many talented peers, and all around fun people, who have helped to keep my mind active and my heart light.

Special thanks to my advisor, Travis Wheeler, who believed in me from the very start; who challenged me to do good science and treated me with compassion. I could not have asked for an better mentor.

This work was supported by grants from the DOE (DE-SC0021216) and the NIH (NIH 1R15HG009570-01).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

Pairwise sequence alignment (Fig. 1.1) is the dominant mechanism for the annotation of protein sequences. Databases of previously annotated proteins (e.g. Uniprot [1]) or protein domains (e.g. Pfam [2]) can be used to classify an unidentified sequence by aligning the new sequence to each of the database sequences, seeking evidence of evolutionary relationship. The term used for these relationships is homology, and it suggests the existence of a common ancestor and, often, a common function. For each alignment, we begin with the null hypothesis that the unidentified sequence (which we will call the target sequence) and the annotated sequence from the database (which we will call the query sequence) are not related (non-homologous). If the queries from one protein family produce particularly high quality alignments with a target sequence, we may be able to reject the null hypothesis and label the target as belonging to that family.

To test the null hypothesis, we need both a model of homology and a model of non-homology, called the null model. These models can be as simple as saying that homologies should have more matches than mismatches or gaps. An alignment can then be assessed under this model by adding one point for each match and subtracting one for each mismatch or gap. If the final sum is positive

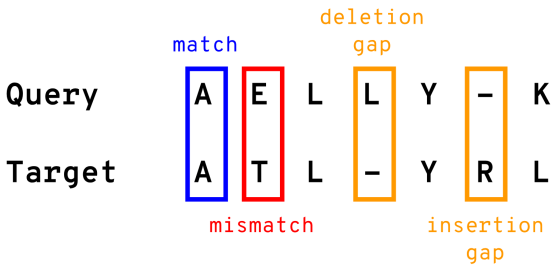


Figure 1.1: Pairwise sequence alignments are performed between two sequences, or between one sequences and one sequence model. The alignment organizes letters from the sequences into columns. Each column can consists of one of three types of pairings. In a match column, highlighted above in blue, two identical letters are paired. In a mismatch column, highlighted above in red, two non-identical letters are paired. In a gap column, highlighted above in orange, a letter from one sequence is paired with a gap character. When the gap character is in the target it is called a deletion, when it is in the query it is called an insertion. If the two aligned sequences are indeed related, then the mismatches and gaps represent mutations where letters have been substituted, removed or added.

then it supports homology, but if it is negative the null model is more likely. This model works for sequences that are so closely related as to be mostly identical, but not for those separated by more evolutionary distance [3]. More sophisticated models, called substitution matrices (Fig. 1.2), were built based on large sets of trusted alignments between homologous sequences. The substitution matrix gives each pair of amino acids its own score. These scores correspond to the ratio of the probability finding that pair of letters in an alignment of two related proteins (homology) over the probability of finding those two amino acids by randomly selecting two letters (null model).

Substitution matrices, such as BLOSUM [6] and PAM [7], are used by many sequence alignment tools, including the popular software BLAST [8]. However, by using a fixed scoring scheme

BLOSUM62

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4																			
R	-1	5																		
N	-2	0	6																	
D	-2	-2	1	6																
C	0	-3	-3	-3	9															
Q	-1	1	0	0	-3	5														
E	-1	0	0	2	-4	2	5													
G	0	-2	0	-1	-3	-2	-2	6												
H	-2	0	1	-1	-3	0	0	-2	8											
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Figure 1.2: Substitution matrices include an integer score for each pair of amino acids. Above is the BLOSUM62 matrix [4], which was constructed from a dataset of homologous alignments of sequences that are at least 62% identical. The positive scores are highlighted in red to show that substitutions do not always count against homology. The score (S) of each pair of letters (i and j) is calculated as a rounded log odds ratio [5]: $S_{ij} = lg(\frac{q_{ij}}{(p_i * p_j)}) * \frac{1}{\lambda}$ Here, p_i and p_j represent the probability of any letter in any protein from the dataset being an i or a j, and q_{ij} is the probability of any column in any homologous alignment in the dataset containing the pair (i,j). The ratio gives us the probability of seeing i and j aligned in a homology divided by the probability of seeing i and j in any two sequence. To produce integer values, probabilities are converted into log space, modified with a scaling factor λ, then rounded to the nearest integer. Insertions and deletions (*indels*) require gap scores (the cost of aligning a letter to a gap character). Most schemes utilize affine gap penalties, in which the first gap character, "gap open", is more costly than than any consecutive gap characters, "gap extensions".

regardless of the specific proteins being aligned, these programs miss out on nuanced amino acid conservation patterns specific to each protein family. Profile hidden Markov models (pHMMs) can be used to model a collection of one or more related proteins, representing the family with a scoring scheme that reflects position-specific patterns for both substitutions and gaps. The sequence alignment tools within the HMMER software package model homology by building pHMMs from multiple sequence alignments (MSAs) of related query sequences (Fig. 1.3), and test the null hypothesis by comparing them to a simple null pHMM (R) (Fig. 1.4). This approach provides superior sensitivity in protein homology search [9].

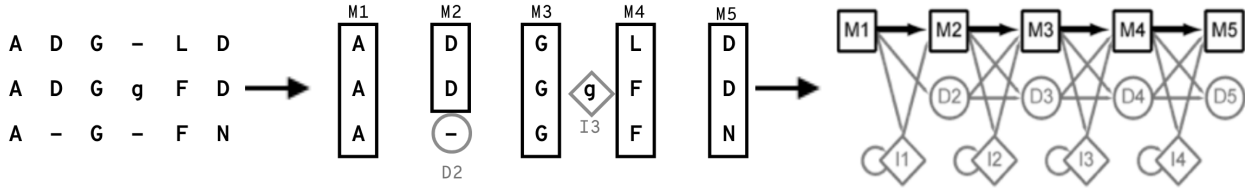


Figure 1.3: Above we see an illustration of how HMMER turns an MSA of homologous sequences into the three core states, match (M), insert (I) and delete (D), of a protein pHMM. Each M state corresponds to one column in the MSA and has a set of 20 emissions probabilities, one for each amino acid, which are analogous to substitution scores. These probabilities are drawn from the distribution of amino acids observed in the corresponding MSA column. To prevent over-fitting, the MSA distributions are combined with a background distribution learned from a large set of protein families [10]. Transition probabilities give the likelihood of moving from one state to another and are analogous to gap penalties. Insertions or deletions observed in the MSA (such as at D_2 and I_3 above) mix with background indel probabilities to yield position-specific transition probabilities. This includes both the gap open transitions from an M state to an I or D state, and the gap extension transitions from I_j to I_j or from D_j to D_{j+1} [11, 12].

Profile HMMs are generative models. Given that the model (Y) contains the set of probabilities defining a protein family, it can be used to compute the probability that a given sequence (X) would be generated by the model: P(X|Y) [11]. To find P(X|Y), we must consider all the possible

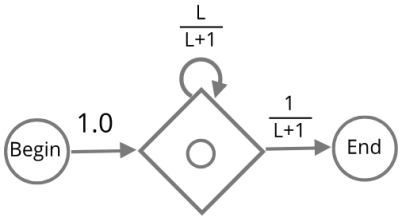
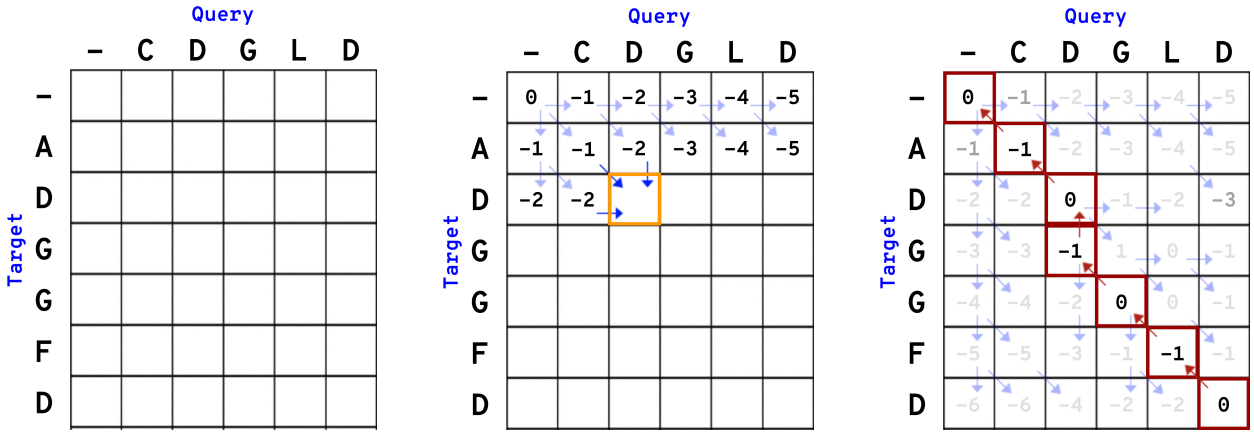


Figure 1.4: In HMMER the null model provides the denominator for the likelihood ratio used to determine if the null hypothesis can be rejected. For protein target sequences, the null model emits amino acids according to their background frequencies. The null model sets it's transition probabilities according to a geometric length distribution so that the expected length of sequence it emits will equal the length (L) of the target [13].

the alignment between the target sequence and the query model (the path Z through the model Y to generate X), but it is not possible to simply enumerate them, since there are approximately $\frac{2^{2N}}{\sqrt{2\pi N}}$ such alignments (where N is the length of the sequences) [14]. Fortunately, a class of dynamic programming (DP) algorithms addresses this limitation. The classic Viterbi [15] algorithm computes a maximum-probability path Z, with run time $O(L * M)$, where L is the length of the target and M is the length of the query. A simple example of dynamic programming for sequence alignment is provided in Figure 1.5.



(a) Empty DP matrix

(b) Memoizing Unrolled Recursion

(c) DP Traceback

$$S_{i,j} = MAX \left\{ \begin{array}{l} S_{i-1,j-1} + Substitution(i,j), \\ S_{i-1,j} + GapCost, \\ S_{i,j-1} + GapCost \end{array} \right\}$$

(d) DP Recursion

C	D	–	G	L	D
A	D	G	G	F	D
-1	+1	-1	+1	-1	+1

(e) Recovered Global Alignment

Figure 1.5: Dynamic Programming (DP) algorithms for sequence alignment (see [14]) enable computation of the optimal alignment score (S) for aligning query to target. In (d) we see a simple DP recurrence for finding the best possible score of aligning the first i letters from the target to the first j positions from the query. To find the best score over the full length of sequences, we simply seek the value of $S_{L,M}$, where L is the length of the target and M is the length of the query. In (a) we see an example of an L x M DP matrix used to store memoized solutions to subproblems. To fill this matrix, computation of the recurrence is performed top-down, observing that each cell can be solved once the cells directly above and to the left have been solved. In (b) we see that the cell highlighted in orange can be solved by applying the recurrence from (d). In this case I have used a scoring scheme of +1 for matches and -1 for mismatches or gaps. Once the matrix has been filled, a traceback is used to recover the path that leads to the optimal alignment score, as seen in (c). The path through the highlighted cells is equivalent to the columns in the alignment, which we see in (e).

The examples so far have focused on proteins, but sequence alignment is also be used for the annotation of DNA. However, in the case of protein-coding DNA, comparison at the level of the encoded amino acids (so-called 'translated alignment') generally produces more accurate alignments [16, 17]. DNA encodes proteins through the use of codons, in which nucleotide triplets encode amino acids according to one of many translation mappings. The mapping seen in eukaryotes is shown in the codon translation table by Figure 1.6.

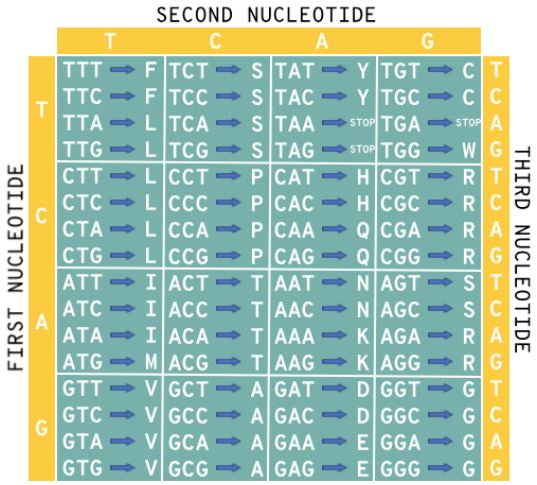
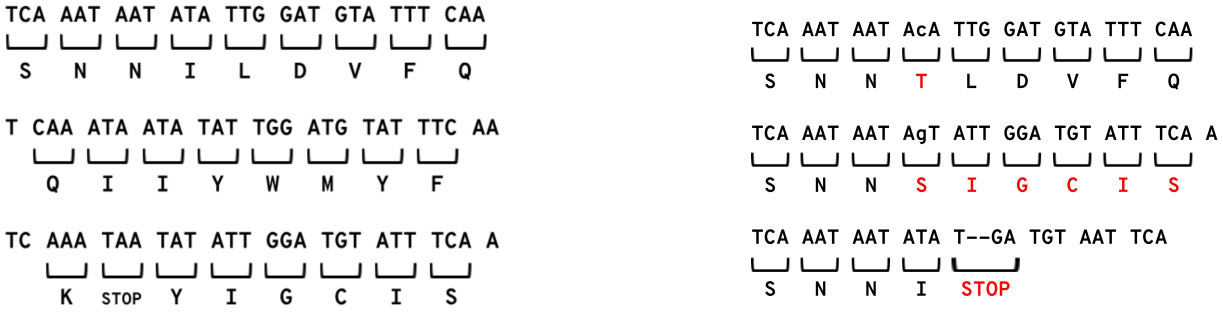


Figure 1.6: Protein coding DNA is made up of consecutive codons. Each codon is three nucleotides long and maps either to one of the twenty amino acids or to a stop codon (which terminates a growing protein). Open reading frames (ORFs) are runs of amino-acid encoding codons containing no stop codons.

One approach to translated alignment is to perform the translation of the target DNA up front, converting codons into amino acids in all six frames (three from each of the DNA's stands) (Fig. 1.7a). The resulting collection of open reading frames (ORFs) can then be searched against a query protein database. This method is used by the translated alignment tools in both HMMER (hmmsearcht) and BLAST (tblastn). However, upfront translation only works if the codons can be translated into the correct amino acids. When a DNA sequence contains insertions or deletions (indels) that disrupt the three-letter codon period (Fig. 1.7b), this strategy can fail to find sequence relationships.

Frameshifts can occur naturally, particularly in pseudogenes (sequences that once encoded functional protein, but have since become inactive and allowed to accumulate random mutations). Frameshifts can also occur as the result of sequencing technology error. The latter is particularly common with the newer long-read sequencers such as PACBIO and Oxford NanoPore[18]. If the proper translation of these sequences cannot be found, translated alignment will fail.

Here I present FATHMM (Frameshift Aware Translated Hidden Markov Models), a tool designed to perform translated alignment on sequences that may contain frameshifts. While other



(a) Three Frame Translation

(b) Translation Errors

Figure 1.7: Each strand of DNA can be translated in three separate frames, with an additional three on the reverse complement strand. In (a) we see the translation of three different frames in the same DNA sequence. Since we cannot be sure which frames are used to make proteins, translated search tools attempt to align all six. In (b) we see the impact of errors in the DNA on translation. The top translation has a substitution error. This changes the translation of one amino acid but does not impact the rest of the translation. In the bottom two translations, indels cause frameshift errors by changing the start and end points of each subsequent codon. This can cause the rest of the sequence to be translated out of frame or to be cut short by the introduction of a stop codon.

tools have been developed to address frameshifts, including FASTA[19], LAST[20] and GeneWise[21], FATHMM's approach is unique. By creating a modified pHMM model and novel implementations of DP algorithms, FATHMM is able to produce more accurate translated alignments for sequences that are both highly divergent and heavily frameshifted. FATHMM is available on github, in the frameshift branch of HMMER, at https://github.com/TravisWheelerLab/hmmer/tree/frameshift.

CHAPTER 2 Methods

FATHMM was built within the HMMER code base. The alignment tools inside of HMMER all rely on a basic pipeline design consisting of four parts, Preprocessing, Filtering, Domain Definition, and Output Generation. FATHMM uses this same general architecture for its pipeline, with several key innovations. Figure 2.1 compares the FATHMM pipeline to that of its closest HMMER relative, the translated search tool hmmsearcht which is currently available via a pull request from the TravisWheelerLab fork of the main HMMER github repo, https://github.com/TravisWheelerLab/hmmer/tree/frameshift).

2.1 Preprocessing

Both FATHMM and hmmsearcht require two input files. The first file contains the target DNA sequence(s) in a recognized format, such as fasta. The second is an HMM query file that contains one or more protein pHMMs. A pHMM file can be generated from an MSA file by another HMMER program, hmmbuild. Each pHMM in the file contains basic information (family name, HMM length, etc), a set of emissions and transition probabilities for each position in the model, and learned parameters for determining the significance of the alignment scores produced by the DP algorithms. HMMs produced with FATHMM's modified hmmbuild tool include an extra set of statistical parameters used to asses the significance of scores produced by frameshift-aware DP algorithms. FATHMM pHMMs also include a MAXL field. The MAXL value is an estimate of the maximum length of a sequence which could belong to the family represented by the pHMM - specifically, it is the length L such that only $1 * 10^{-7}$ of the sequences emitted by the model are expected to be longer than L. HMM files built in other versions of HMMER will not include these

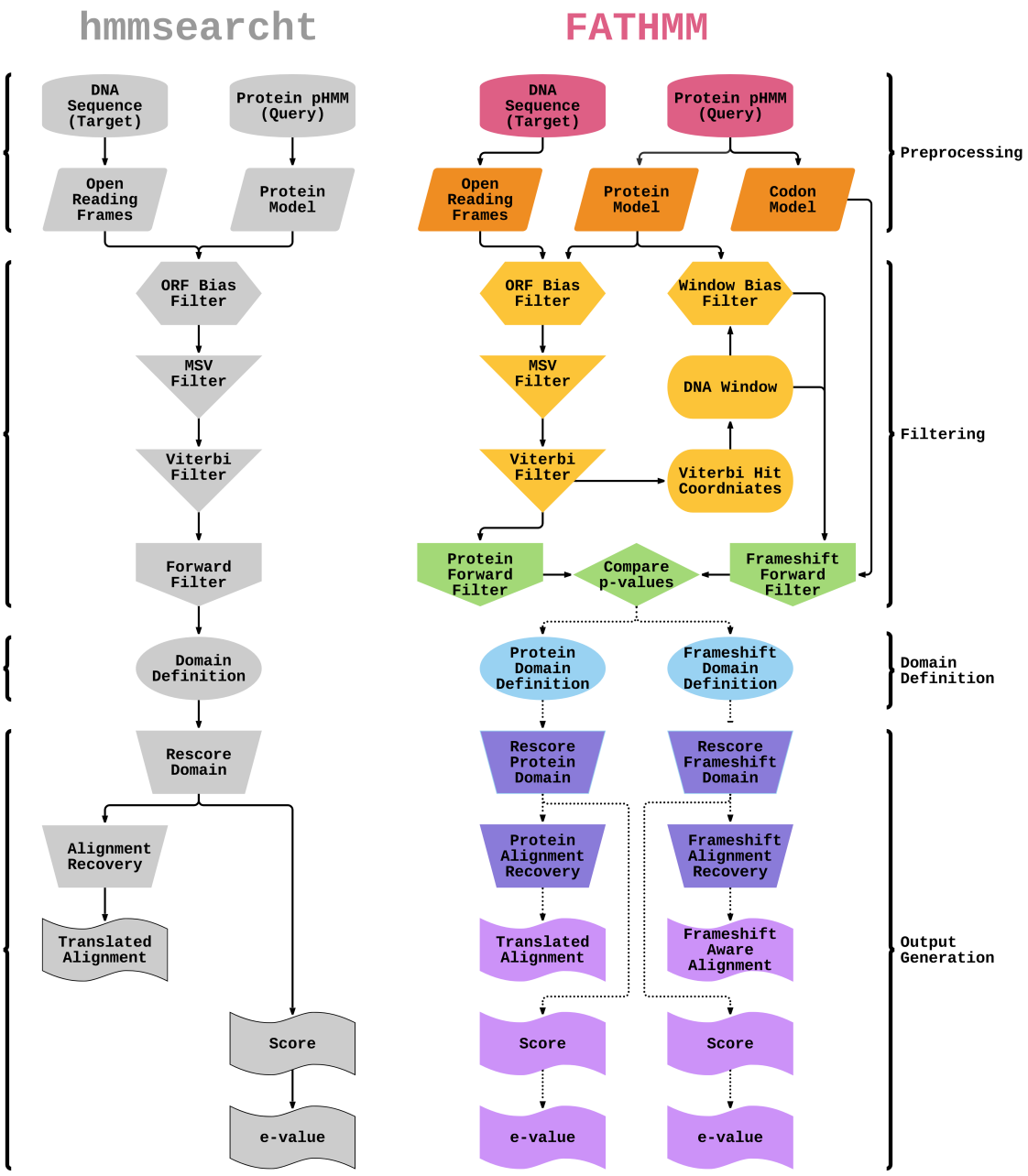


Figure 2.1: Side by side comparison of the pipelines for FATHMM (shown in color) and hmmsearcht (shown in grey). The pipelines are separated into 4 stages: (1) Preprocessing, (2) Filtering, (3) Domain Definition, and (4) Output Generation. During Preprocessing, the input files are read and their data is converted and stored. In the Filtering stage, three DP algorithms are run in a series, with each estimating alignment quality and removing non-homologous target-query pairs. For target-query pairs that make it past Filtering, the potential areas of homology are isolated and analyzed during the Domain Definition stage. Finally, Output Generation produces an alignment, score, and E-value for each detected homology.

variables and must be passed though the hmmconvert tool before running FATHMM.

The Preprocessing step consists of converting the textual information in these input files into data structures that can be efficiently accessed by the relevant subroutines. For both hmmsearcht and FATHMM, this includes translating each target DNA sequence into the set of all possible ORFs on both the input strand and its reverse complement. Translation is done via a hard coded codon translation table, and creates an ORF for every stretch of at least 20 amino acids that is not interrupted by a stop codon. To accommodate organism-specific codon usage, the user may specify a codon translation table with the -c flag. The default is the codon translation utilized by most eukaryotes.

The emission and transition probabilities from the HMM file are stored as logarithm-transformed floats. As with the BLOSUM62 matrix (Fig 1.2), the emission probabilities are used as the numerators of log odds ratios, by dividing the probability of a homologous substitution by the probability of a random pairing. At each match state (M_j) in the pHMM, the score for each of twenty amino acids, x_i, is computed as the log of the ratio of the match state's emission probability, $P(M_j(x_i))$, and its null model probability, $R(x_i)$, (Eq. 2.1). This yields a positive score if the pairing supports homology and a negative score if does not. The null transition probabilities, $R(T)$, depend on the length of the target sequence, and are introduced later during the Filtering and Domain Definition Stages. For now, the probability of transition from one state to another, say from M_j to M_{j+1}, is simply stored as its log (Eq. 2.2).

$$E(M_j(x_i)) = log_2(\frac{P(M_j(x_i))}{R(x_i)}) \tag{2.1}$$

$$T(M_j, M_{j+1}) = log_2(P(M_j, M_{j+1}) \tag{2.2}$$

Once the emission and transition scores for the core model have been calculated, hmmsearcht and FATHMM both include additional states, which I will refer to as special states, to the beginning and end of the model (Fig. 2.2. These states allow for local alignments by emitting non-homologous portions of the target before, after, and between any homologous sections. They emit amino acids

at the same background distribution as the null model and use similar expected length distributions to determine their transition probabilities.

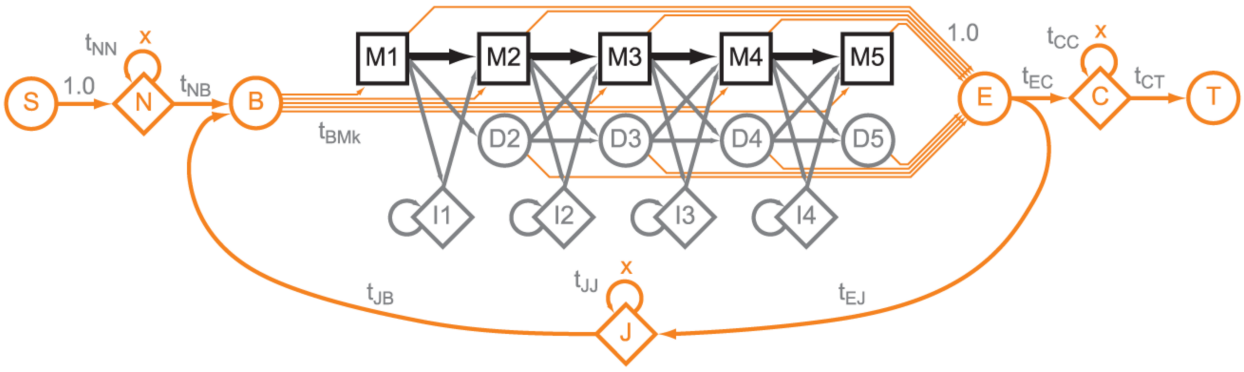
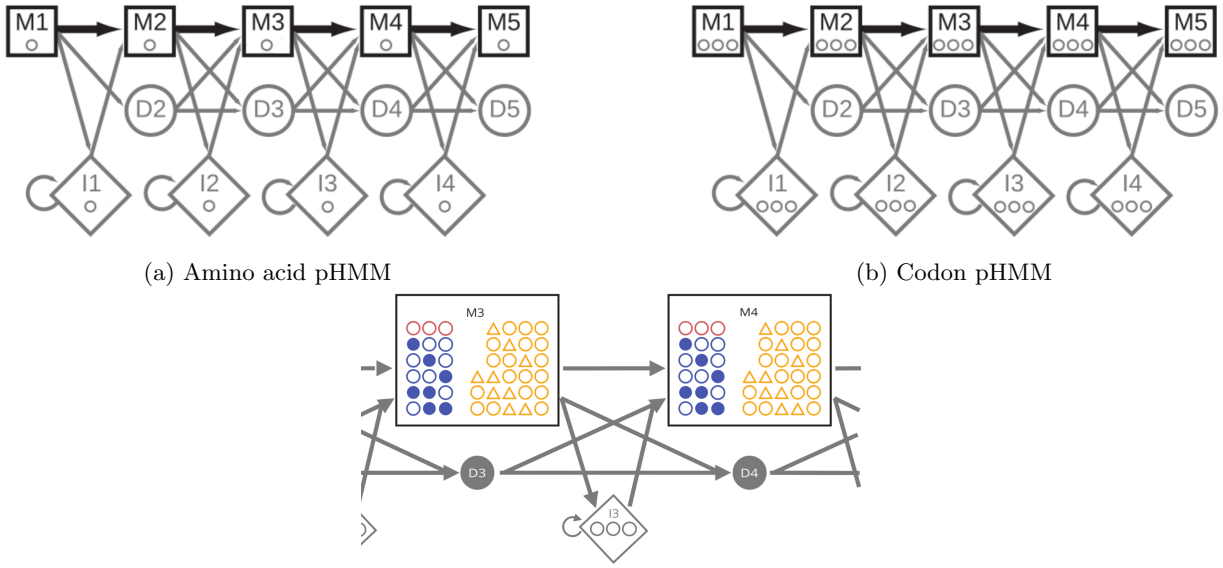


Figure 2.2: To allow for local alignment between the target and query, HMMER uses a pHMM with a core model, consisting of the match, insert and delete states (M, I, and D), and several special states. I describe the model briefly here - see [13] for further details. All paths through the pHMM begin at the S state and then transition the to N state. The N state emits residues from the target (e.g. amino acids or nucleotides) which come before the area of homology with the query. The N state transition probabilities ($\frac{L}{L+3}$ for N to N transitions and $\frac{3}{L+3}$ for N to B transitions) depend on the the length of the target (L). This yields an expected $\frac{L}{3}$ length of sequence emitted by N. The B state allows for the alignment between target and query to begin at any M state in the core model. The alignment can then exit the core model from that M state, or any subsequent M or D state, by transitioning to the E state. Since a local alignment through a length M model may include any of $\frac{M(M+1)}{2}$ possible start/end pairs, the uniform local alignment fragment length distribution of $\frac{2}{M(M+1)}$ is used as the transition probability from the B state to any M state. The E state can then transition to either the J or the C state. The J state emits residues and then reenters the core model via the B state, allowing for a target to contain multiple disconnected regions of homology. The C state emits residues that come after the last homologous region has been emitted, and then transitions to the T state at the end of the target sequence. Both the J and C states have the same transition probabilities as the N state, accounting for the other $\frac{2L}{3}$ of the sequence length.

In addition to the reading and adjusting the protein pHMM, FATHMM creates a Frameshift Aware (FA) codon model (Fig. 2.3) based on the protein model. A codon model without frameshift awareness would simply replace the match state amino acid emissions probabilities with codon emissions probabilities, according to the codon translation table, while also including some way of handling stop codons. Transition probabilities would remain the same but the I and D states would pertain the the insertion or deletion of codons, rather than amino acids.To adding frameshift awareness, the model must also include some way to account for the possibility of individually inserted or deleted nucleotides.

One approach might be to create an additional state that allows the path through the model



(a) Amino acid pHMM

(b) Codon pHMM

(c) Frameshift aware codon pHMM

Figure 2.3: FATHMM uses both a protein pHMM and a frameshift aware codon pHMM. In the protein pHMM, shown in (a), both the match states (M) and insert states (I) emit a single amino acid. A codon model without any frameshift awareness, as shown in (b), would emit three nucleotides from its match and insert states. A zoomed in view of the frameshift aware model is seen in (c). For this model, the insert state is identical to (b) but the match states emits a variable length string of nucleotides. These include both codons (shown as three pink circles) and pseudo-codons. Pseudo-codons with fewer than three nucleotides (show in blue) are presumed to be missing deleted nucleotides (shown as darkened circles). Pseudo-codons with more than three nucleotides (shown in yellow) are presumed to contain inserted nucleotides (shown at triangles).

to move between frames by emitting either one or two nucleotides between codons. I considered this strategy, but determined that integrating frameshifts into the match state emission provided greater flexibility in modeling the length and location of the indels and retained any evolutionary information present in the original nucleotides. To accomplish this, the FA model match states can emit not only codons, which are always three nucleotides long, but also 'pseudo-codons' that contain indels and can be of length 1, 2, 4 or 5 nucleotides.

To create the FA model, first all the codons that do not encode a stop are mapped to their amino acid translations and the corresponding emission scores, using a codon translation table (Fig. 1.6). For instance, the codon CAA would be mapped to the amino acid Q using the translation table shown in Figure 1.6 and the probability that a match state in the protein model had assigned to emitting Q would be assigned to the emission of the codon CAA by the corresponding match state

in the FA model. The model also needs to be able to accommodate stop codons, which can arise in protein coding DNA due to substitution or frameshift errors; this is done via a 'substitute one' max function in which each stop codon is mapped to the maximum scoring codon that can be made by substituting one of its three nucleotides. Since the maximum scoring codon was already mapped to it's amino acid translation this becomes the stop codon translation as well (Fig. 2.4). For instance, by substitution one of its nucleotides, the stop codon TAA and be changed to one of nine other possible codons, which in turn can be translated into 6 different amino acids. Whichever of those six amino acids has the highest emissions probability at a match state in the protein model will be mapped to the stop codon for the corresponding match state in the FA model.

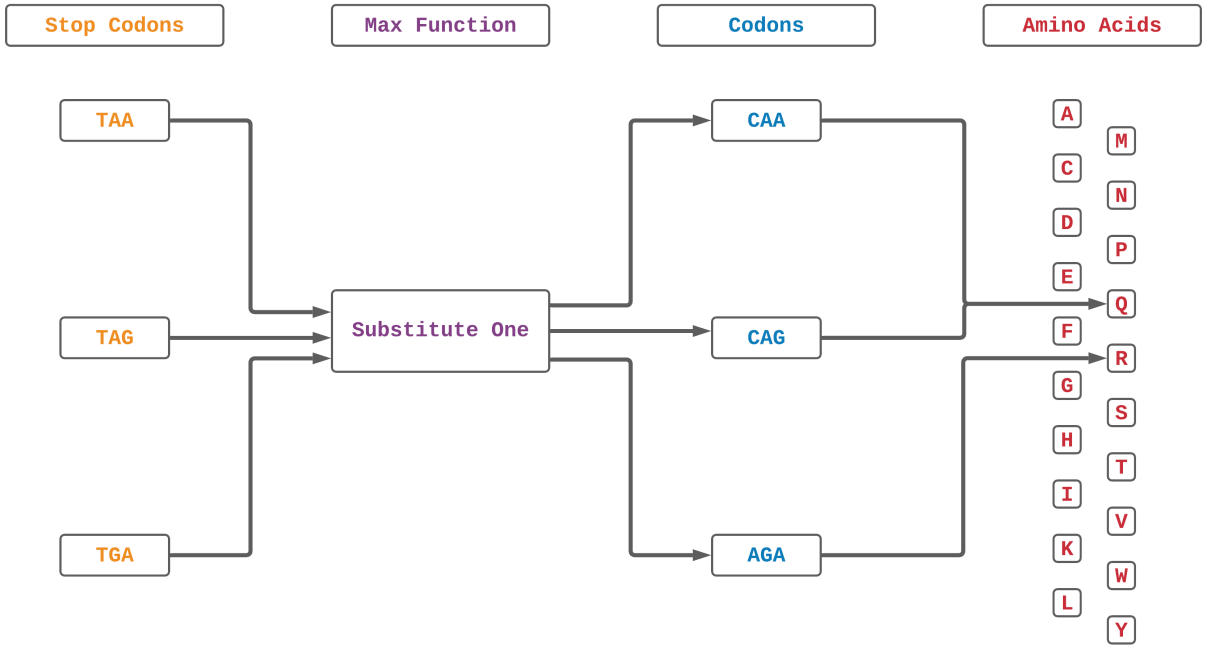


Figure 2.4: A pictorial example of how stop codons are mapped to codons in the FA codon model. Using the standard eukaryotic codon translation table, TAA, TAG, and TGA all encode a stop. The protein model match state in this example has Q as its highest scoring amino acid emission, followed by E and then R. Two of the stop codons can be changed to codons that translate to Q by substituting a single nucleotide. For the third stop codon the highest scoring codon it can be changed to with a single substitution translates to R. The stop codons must also pay a substitution penalty s, which by default is 0.01. This means that, for our example, $E(M_j(TAA))$ in the FA codon model is equal to $E(M_j(Q))$ from the protein model, plus $log2(s)$. Once all stop codons are mapped, the other non-stop codons have their emissions scores normalized so that, for instance, $E(M_j(CAG)) = E(M_j(Q)) + log2(1-s)$.

Once all the codons and stop codons have been mapped, the model moves on to mapping the

pseudo-codons (Fig. 2.5). This is done by inserting or deleting nucleotides to form proper three nucleotide codons and mapping the pseudo-codon to the one with the highest score. As with the stop codons, this also assigns the mapped codon's amino acid translation to the pseudo-codon. To prevent the model from emitting stop codons and pseudo-codons erroneously, a low probability is assigned to these emissions. The user can adjust this probability with the --fs option which is defaulted to 0.01 for stop codons and for pseudo-codons with a single nucleotide indel, and to 0.005 for pseudo-codons with two nucleotide indels.

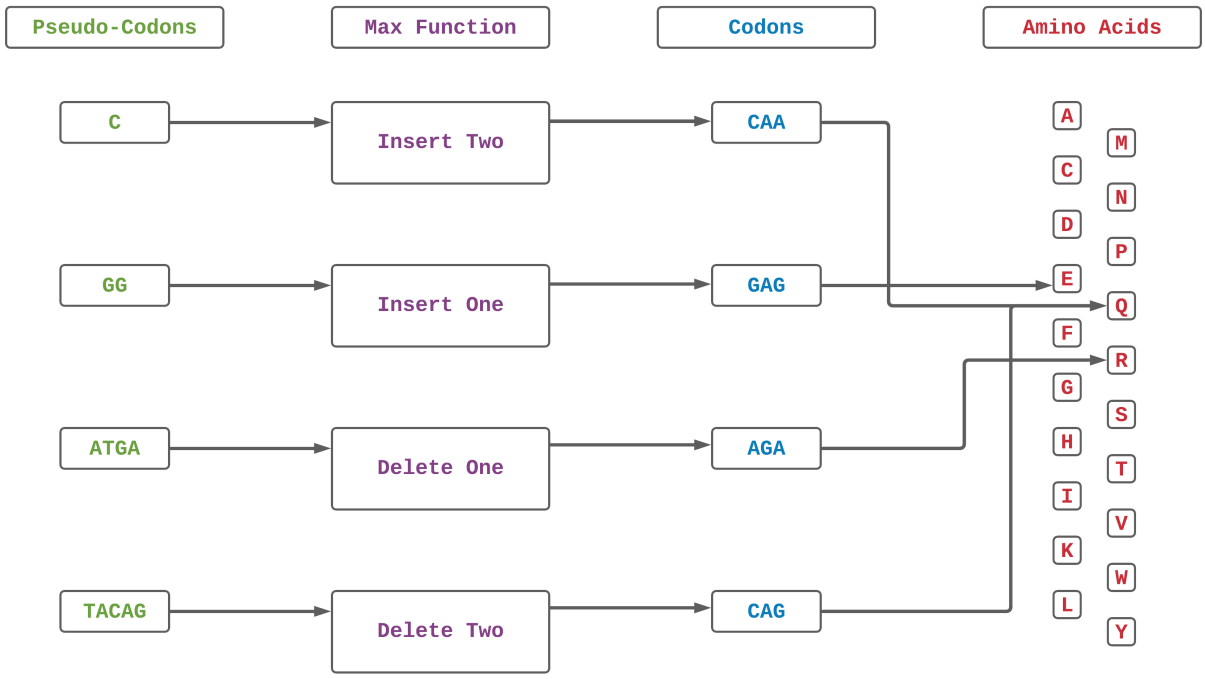


Figure 2.5: An example of how the four types of pseudo-codons are mapped to codons. For this example we have a match state where the top amino acid emissions in the protein model are Q, then E, then R. One-nucleotide pseudo-codons require two nucleotides to be inserted to find the maximum scoring codon that can be made. In our example the addition of two A's allows the pseudo-codon C to be mapped to a codon that translates to Q, which is the highest scoring of all alternatives. For two-nucleotide pseudo-codons, the same is done but with just a single nucleotide insertion. For the four-nucleotide pseudo-codons, one of the nucleotides must be deleted to make a codon. In our example, deleting the T in ATGA allows it to map the highest possible score as the codon AGA, which translates to E. The same is done for five-nucleotide pseudo-codons, but with two nucleotides being deleted. A frameshift cost, f, is applied to all pseudo-codons, and the codon emissions score are normalized with he remaining probability. By default $f = 0.01$ and the cost for a pseudo-codon with z inserted or deleted nucleotides is $log_2(\frac{f}{z})$. This gives each match state in the FA model a 1% probability each of emitting a two or four nucleotide pseudo-codon, a 0.5% each of emitting a one or five nucleotide pseudo-codon and a 97% percent chance of emitting a codon.

Since each match state in the protein model has its own set of amino acid emissions scores, this mapping must be done once for each position in the model, increasing the model's memory usage. The protein model uses a 20*M float matrix to store the emissions scores of the 20 amino acids, (where M is the length of the model). The FA codon model has 64 codons and and 1300 pseudo-codons. This means the space to store the emissions increases to a 1364*M float matrix. This surely has an impact on cache performance when the FA model in used in DP algorithms.

Finally, the special states are added to the FA codon model. In the protein model the N, J and C states all emit single amino acid. If the FA model special state were to emit single nucleotides this would allow for an alignment to exit the core model in one frame, emit one or two nucleotides from the J state, and then reenter the core model in a different frame. By having the N, J and C states emit codons instead of single nucleotides the FA model ensures that the only way a path can move from one frame to another is by emitting a pseudo-codon from a match state. This required the creation of two additional sets of special states (see Fig. 2.6) so that each set could operate in a single frame.

The Prepossessing step concludes when the data in the two input files has been read, converted, and stored in memory. This includes the target as both a DNA sequence and its translated ORFs, as well as the query as a protein pHMM and, in the case of FATHMM, an FA codon model. Both hmmsearcht and FATHMM then begin to consider the potential for homology between targets and queries.

2.2 Filtering

The Filtering stages of hmmsearcht and FATHMM are designed to decrease the runtime of these programs. They do this by filtering out the majority of non-homologous pairs with a series of increasingly stringent (and slow) DP algorithms. The filters in hmmsearcht use the protein pHMM as the query and translated ORFs as the targets. The same is true for the first three DP filters in FATHMM, with the codon model and DNA sequence only being used by an additional fourth DP filter. The decision to use ORFs for these filters in FATHMM, despite their potential

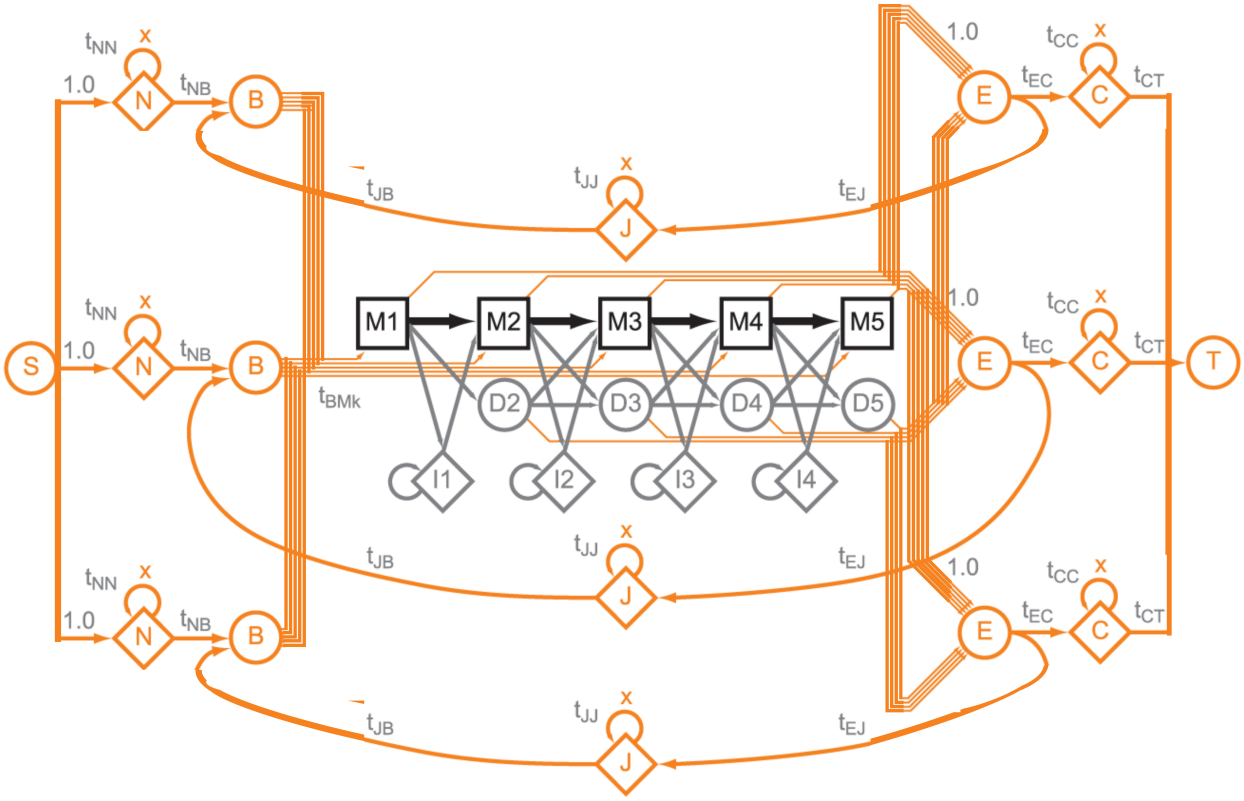


Figure 2.6: Each frame of the target DNA is handled by its own set of special states. The N, J, and C states each emit codons from just one frame. This means that the only way for probabilities from one frame to be included in the probabilities of another frame is through the pseudo-codons. The transition probabilities for these states are the same as for the protein pHMM, except that each set of specials is only responsible for the $\frac{1}{3}$ of L. For instance, the N to N transitions are changed form $log_2(\frac{L}{L+3})$ to $log_2(\frac{\frac{L}{3}}{\frac{L}{3}+3})$.

for mistranslation due to frameshifts, was based on the experience that a sequence that is too heavily frameshifted to produce an ORF capable of passing these filters is typically also too heavily frameshifted to be recovered at all, even with a frameshift model. In the current implementation, these filters are unmodified from those outlined in [9], but future work may include adding frameshift awareness to these filtering steps. I describe them here for completeness.

The first filter is a DP algorithm called multiple segment Viterbi (MSV) (Fig. 2.7). This algorithm looks for short sections of ungapped alignment by using only the match states of the core pHMM. The score produced by MSV includes the emissions and transitions probabilities for the maximum scoring ungapped alignments between the target and query, as well as the null emission

Multiple Segment Viterbi (MSV)

/* Special conditions for 0^{th}, L^{th} and M^{th} indices are omitted for simplicity */

for *i in* $\{1..L\}$ **do** // For each residue in the target

$N(i) \leftarrow N(i-1) + T(N,N)$; // N state

$B(i) \leftarrow MAX \left\{ \begin{array}{l} N(i) + T(N,B), \\ J(i) + T(J,B) \end{array} \right\}$; // B state

for *j in* $\{1..M\}$ **do** // For each position in the core model

$M(i,j) \leftarrow MAX \left\{ \begin{array}{l} B(i-1) + T(B,M), \\ M(i-1,j-1) + T(M_{j-1}, M_j) \end{array} \right\} + E(M_j(x_i))$; // M state

$E(i) \leftarrow MAX \left\{ \begin{array}{l} M(i,j) \\ E(i) \end{array} \right\}$; // E state

end

$J(i) \leftarrow MAX \left\{ \begin{array}{l} E(i-1) + T(E,J) \\ J(i-1) + T(J,J) \end{array} \right\}$; // J state

$C(i) \leftarrow MAX \left\{ \begin{array}{l} E(i-1) + T(E,C) \\ C(i-1) + T(J,C) \end{array} \right\}$; // C state

end

$T \leftarrow C(L) + T(C,T)$; // T state

Figure 2.7: The Multiple Segment Viterbi (MSV) DP algorithm seeks to identify a high-scoring run of matches between the target and query, containing no indels. It uses a doubly-nested loop to calculate each cell in the $L * (M + 5)$ matrix, where L is the length of the target sequence, M is the length of the query core model, and 5 is the number of special sates which are part of the matrix (N, B, E, J, and C). The outer loop is used for the N, J, C, and B states. The emissions probabilities from these special states are taken from the same background distributions as the null model. Therefore, the log odds ratios used to make their emissions scores always come out to zero and are not included in the calculation. The inner loop is used for the core match states and the E state. The E state, E(i), transitions from whichever M(i,j) state has the highest score. After the loops finish, the T state transitions from the final C state, C(L), and holds the full MSV score.

probabilities built in to the emissions scores. What the score does not yet contain is the null transition probabilities. These are calculated as a single null score (NS) based on the length (L) of the target (Eq. 2.3). By subtracting this log probability from the MSV score, the pHMM transitions probabilities are finally converted to log odds ratios.

$$NS(X) = L * log_2 \left(\frac{L}{L+1} \right) + log_2 \left(\frac{1}{L+1} \right) \tag{2.3}$$

Some sequences can have a highly skewed distribution of residues, leading to false positives between repetitive sections of the target and query [9]. To prevent this, a heuristic DP algorithm is used to compute a bias score (BS) (Eq. 2.4), which can be used to adjust the default null emissions

score to account for a difference between the background distribution and the biased distribution. When subtracted from the alignment score, the BS effectively replaces the old background distribution null model with a new null model informed by the distribution in the pHMM's consensus sequence (Eq. 2.5). If the old null model overestimated the probability of non-homology, the BS will be negative and the alignment score will increase. If non-homology was underestimated, the BS will be positive and the alignment score will decrease.

$$BS(x_i) = log_2\left(\frac{newR(x_i)}{oldR(x_i)}\right) \tag{2.4}$$

$$E(M_j(x_i)) - BS(x_i) \approx log_2\left(\frac{P(M_j(x_i))}{oldR(x_i)}\right) - log_2\left(\frac{newR(x_i)}{oldR(x_i)}\right) = log_2\left(\frac{P(M_j(x_i))}{newR(x_i)}\right) \tag{2.5}$$

MSV scores are known to follow a Gumbel distribution (Eq. 2.6), where $\lambda \approx e^z$ and z is the log base of $S(X,Y)$ [9]. In our case $z = 2$, while μ, being specific to each pHMM, was determined by hmmbuild and is thus read from the HMM file. Once the MSV alignment score t has been adjusted by the null and bias scores, the K and λ computed by hmmbuild are used to find the p-value $P(S(X,Y) > t)$ - the probability that an MSV score of t or higher would be produced by the query and a randomly generated sequence. Both hmmsearcht and FATHMM discard any target-query pair with an MSV p-value over 0.02, removing an expected 98% of all non-homologous pairs. The rest are passed on the second filter.

$$P(S(X,Y) > t) \approx 1 - e^{-e^{-\lambda(t-\mu)}} \tag{2.6}$$

The second filter is the Viterbi DP algorithm (Fig. 2.8). Using all three states in the core pHMM, it finds the maximum scoring alignment between the target and query, allowing for insertions and deletions. The scores it produces are adjusted by the null and bias scores and then converted to p-values, following the same Gumbel distribution as MSV but with a different value of μ. Only target-query pairs with a Viterbi p-value at or below 0.001 are passed on to the third filter.

Viterbi

/* Special conditions for 0^{th}, L^{th} and M^{th} indices are omitted for simplicity */

1 **for** *i in* {*1..L*} **do** // For each residue in the target

2 $N(i) \leftarrow N(i-1) + T(N,N)$; // N state

3 $B(i) \leftarrow MAX \left\{ \begin{array}{l} N(i) + T(N,B), \\ J(i) + T(J,B) \end{array} \right\}$; // B state

4 **for** *j in* {*1..M*} **do** // For each position in the core model

5 $M(i,j) \leftarrow MAX \left\{ \begin{array}{l} B(i-1) + T(B,M), \\ M(i-1,j-1) + T(M_{j-1},M_j) \\ I(i-1,j-1) + T(I_{j-1},M_j) \\ D(i-1,j-1) + T(D_{j-1},M_j) \end{array} \right\} + E(M_j(x_i))$; // M state

6 $I(i,j) \leftarrow MAX \left\{ \begin{array}{l} M(i-1,j) + T(M_j,I_j) \\ I(i-1,j) + T(I_j,I_j) \end{array} \right\}$; // I state

7 $D(i,j) \leftarrow MAX \left\{ \begin{array}{l} M(i,j-1) + T(M_{j-1},D_j) \\ D(i,j-1) + T(D_{j-1},D_j) \end{array} \right\}$; // D state

8 $E(i) \leftarrow MAX \left\{ \begin{array}{l} M(i,j) \\ D(i,j) \\ E(i) \end{array} \right\}$; // E state

9 **end**

10 $J(i) \leftarrow MAX \left\{ \begin{array}{l} E(i) + T(E,J) \\ J(i-1) + T(J,J) \end{array} \right\}$; // J state

11 $C(i) \leftarrow MAX \left\{ \begin{array}{l} E(i) + T(E,C) \\ C(i-1) + T(J,C) \end{array} \right\}$; // C state

12 **end**

13 $T \leftarrow C(L) + T(C,T)$; // T state

Figure 2.8: The Viterbi DP algorithm is very similar to MSV, except that it uses the I and D states to allow for gaped alignments. Insert states, like the N, J, and C states, emit at the background distribution probabilities and so do not need to include an emissions score. Each i,j pairing in the core matrix is captured in its own M, I, and D cells, increasing the size of the matrix to $L * (3M + 5)$.

The third filter, the Forward algorithm (Fig. 2.9), unlocks the full potential of the pHMMs. As stated in the introduction, the fully probabilistic nature of pHMMs enables the calculation of probability that the target (X) would be generated by the query model: $P(X|Y)$. Since there are multiple paths (or alignments) by which Y can generate X, $P(X|Y)$ is found by integrating over (summing) the probabilities of all alignments. Viterbi approximates $P(X|Y)$ by finding the probability of a single optimal path (Z) taken through the model to generate X, $P(X, Z|Y)$. However, the maximum probability path may not be a good approximation of the full $P(X|Y)$, particularly

Forward

/* Special conditions for 0^{th}, L^{th} and M^{th} indices are omitted for simplicity */

for *i in* {*1..L*} **do** // For each residue in the target

$N(i) \leftarrow N(i-1) + T(N,N)$; // N state

$B(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{N(i)+T(N,B)} \\ 2^{J(i)+T(J,B)} \end{array} \right\} \right)$; // B state

for *j in* {*1..M*} **do** // For each position in the core model

$M(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{B(i-1)+T(B,M)} \\ 2^{M(i-1,j-1)+T(M_{j-1},M_j)} \\ 2^{I(i-1,j-1)+T(I_{j-1},M_j)} \\ 2^{D(i-1,j-1)+T(D_{j-1},M_j)} \end{array} \right\} \right) + E(M_j(x_i))$; // M state

$I(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{M(i-1,j)+T(M_j,I_j)} \\ 2^{I(i-1,j)+T(I_j,I_j)} \end{array} \right\} \right)$; // I state

$D(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{M(i,j-1)+T(M_{j-1},D_j)} \\ 2^{D(i,j-1)+T(D_{j-1},D_j)} \end{array} \right\} \right)$; // D state

$E(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{M(i,j)} \\ 2^{D(i,j)} \\ 2^{E(i)} \end{array} \right\} \right)$; // E state

end

$J(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{E(i)+T(E,J)} \\ 2^{J(i-1)+T(J,J)} \end{array} \right\} \right)$; // J state

$C(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{E(i)+T(E,C)} \\ 2^{C(i-1)+T(J,C)} \end{array} \right\} \right)$; // C state

end

$T \leftarrow C(L) + T(C,T)$;

Figure 2.9: The Forward DP algorithm uses all the same state transitions as Viterbi, but rather than picking one maximum scoring transition for each cell, Forward sums all possible transitions. Adding probabilities can not be performed when those probabilities are stored in log space, so probabilities are moved out of log space, addition is performed, and then log conversion is performed to ensure acceptable dynamic range.

when the target ans query are more distally related. The Forward algorithm removes the limitations of an approximated $P(X|Y)$ by summing the probability for all possible alignments: $P(X|Y) = \sum_z P(X,z|Y)$ [13].

The score produced by Forward, $S(X|Y) = \frac{P(X|Y)}{P(X|R)}$, is adjusted with the same NS and BS scores as the MSV and Viterbi scores and then fitted to a probability distribution to find the Forward p-value. Unlike MSV and Viterbi, Forward scores do not follow a Gumbel distribution and instead are relatively "fat-tailed". The exponential of this tail converges to the the log base of the score

for so that Equation 2.7 hold for $t > \tau$. τ can be approximated by first fitting Forward scores to a Gumbel distribution and finding t' such that $P(S(X,Y) < t') = 0.04$ and making $\tau = t'$ [13]. Only target-query pairs with a Forward p-value of less than 1e-5 pass this filter.

$$P(S(X,Y) > t) \propto \mu e^{-\lambda(t-\tau)} \tag{2.7}$$

For hmmsearcht the Forward filter marks the end of the filtering stage. FATHMM uses the MSV, Viterbi, and Forward Filters in same way as hmmsearcht, with the protein pHMM as query and the translated ORFs as targets. However, it also runs a novel frameshift-aware version of the Forward filter with the FA codon model as query. For the target, FATHMM borrows the concept of a DNA window from HMMER's DNA alignment tool, nhmmer [22].

DNA windows are sub-sequences of full length DNA targets (often chromosomes), built around a potential area of homology. In nhmmer the areas of potential homology are identified by the DNA-to-DNA equivalent of the MSV filter. In FATHMM these areas are identified by the original DNA coordinates of the ORFs which pass the Viterbi Filter. The window is created by pulling out a sub-sequence around this area whose length approximates the MAXL variable described in the Preprocessing section, so that it will be certain to encompass any reasonably long match to the query. Finally, any overlapping windows are merged. IN FATHMM this means that the DNA window target used by the FA Forward filter may correspond to more than one of the ORF targets used by the standard Forward filter.

In aligning a DNA target to an FA codon model query, the FA Forward algorithm allows for emissions of both in frame codons and between frame pseudo-codons. It does this by adding a third nested loop to count for the five emission lengths (1-5 nucleotides). A naive approach to this would increase the total operations per match state cell from 13 for standard Forward to 65 for FA Forward (Fig. 2.10). Through memoization, the number of operations for each match state cell can be cut to just 27 (Fig. 2.11).

New null and bias scores are also needed for the DNA window. The DNA window NS score is calculated in three separate frames just as with the the FA N, J, and C states. $NS(F)$ is

Frameshift Aware Forward

/* Special conditions for 0–4^{th}, L^{th} and M^{th} indices are omitted for simplicity */

1 **for** i *in* $\{5..L\}$ **do** // For each residue in the target

2 $N(i) \leftarrow N(i-3) + T(N,N)$; // N state

3 $B(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{N(i)+T(N,B)} \\ 2^{J(i)+T(J,B)} \end{array} \right\} \right)$; // B state

4 **for** j *in* $\{1..M\}$ **do** // For each position in the core model

5 **for** c *in* $\{1..5\}$ **do** // For each codon/pseudo-codon length

6 $M(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{M(i,j)} \\ 2^{B(i-c)+T(B,M)} \\ 2^{M(i-c,j-1)+T(M_{j-1},M_j)} \\ 2^{I(i-c,j-1)+T(I_{j-1},M_j)} \\ 2^{D(i-c,j-1)+T(D_{j-1},M_j)} \end{array} \right\} \right) + E_c(M_j(x_i))$; // M state

7 **end**

8 $I(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{M(i-3,j)+T(M_j,I_j)} \\ 2^{I(i-3,j)+T(I_j,I_j)} \end{array} \right\} \right)$; // I state

9 $D(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{M(i,j-1)+T(M_{j-1},D_j)} \\ 2^{D(i,j-1)+T(D_{j-1},D_j)} \end{array} \right\} \right)$; // D state

10 $E(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{M(i,j)} \\ 2^{D(i,j)} \\ 2^{E(i)} \end{array} \right\} \right)$; // E state

11 **end**

12 $J(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{E(i)+T(E,J)} \\ 2^{J(i-3)+T(J,J)} \end{array} \right\} \right)$; // J state

13 $C(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{E(i)+T(E,C)} \\ 2^{C(i-3)+T(J,C)} \end{array} \right\} \right)$; // C state

14 **end**

15 $T \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{C(L-2)} \\ 2^{C(L-1)} \\ 2^{C(L)} \end{array} \right\} \right) + T(C,T)$; // T state

Figure 2.10: FA Forward uses a third loop (lines 5-7) to sum over the five possible (pseudo-)codon lengths. The Emissions scores for all (pseudo-)codons ending with x_i are stored in $E_c(M_j(x_i))$, with c being the length of the (pseudo-)codons. The N,J, and C sates, as well as the insert state, all emit codons and so transition from three rows above rather than just one. The probabilities from the 3 frames in the C state are summed before transitioning to the T state.

responsible for the one third of the target length, $\frac{L}{3}$, in a single frame. The probability in NS(F) is then multiplied by three to reach the full DNA window null score, $NS(X)$ (Eq. 2.8). To account for the DNA window's multiple frames, the BS calculation begins by translating the DNA window into

Memoized Frameshift Aware Forward

/* Special conditions for 0–4^{th}, L^{th} and M^{th} indices are omitted for simplicity */

for i *in* $\{1..L\}$ **do** // For each residue in the target

$N(i) \leftarrow N(i-3) + T(N,N)$; // N state

$B(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{N(i)+T(N,B)} \\ 2^{J(i)+T(J,B)} \end{array} \right\} \right)$; // B state

for j *in* $\{1..M\}$ **do** // For each position in the core model

for c *in* $\{4..1\}$ **do** // For each memorized transition

$Memo(c+1, j) \leftarrow Memo(c, j)$; // Shift memoized transitions up by one row

end

$$Memo(1,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{B(i-1)+T(B,M)} \\ 2^{M(i-1,j-1)+T(M_{j-1},M_j)} \\ 2^{I(i-1,j-1)+T(I_{j-1},M_j)} \\ 2^{D(i-1,j-1)+T(D_{j-1},M_j)} \end{array} \right\} \right) ;$$

/* Calculate new transition and memoize */

$$M(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{Memo(1,j)+E_1(M_j(x_i))} \\ 2^{Memo(2,j)+E_2(M_j(x_i))} \\ 2^{Memo(3,j)+E_3(M_j(x_i))} \\ 2^{Memo(4,j)+E_4(M_j(x_i))} \\ 2^{Memo(5,j)+E_5(M_j(x_i))} \end{array} \right\} \right) ;$$

/* Sum memoized transitions and codon/ pseudo-codon emissions */

$I(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{M(i-3,j)+T(M_j,I_j)} \\ 2^{I(i-3,j)+T(I_j,I_j)} \end{array} \right\} \right)$; // I state

$D(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{M(i,j-1)+T(M_{j-1},D_j)} \\ 2^{D(i,j-1)+T(D_{j-1},D_j)} \end{array} \right\} \right)$; // D state

$E(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{M(i,j)} \\ 2^{D(i,j)} \\ 2^{E(i)} \end{array} \right\} \right)$; // E state

end

$J(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{E(i)+T(E,J)} \\ 2^{J(i-3)+T(J,J)} \end{array} \right\} \right)$; // J state

$C(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{E(i)+T(E,C)} \\ 2^{C(i-3)+T(J,C)} \end{array} \right\} \right)$; // C state

end

$T \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{C(L-2)} \\ 2^{C(L-1)} \\ 2^{C(L)} \end{array} \right\} \right) + T(C,T)$; // T state

Figure 2.11: The transition calculations for the five emissions length of the FA match states can be memoized in a $5 * M$ matrix (Memo). Cell $M(i,j)$ and cell $M(i-1,j)$ transition from the same cells for four out of five emissions lengths. For each $M(i,j)$, Memo(5,j) is equal the (Memo(4,j) for $M(i-1,j)$, Memo(4,j) is equal the Memo(3,j) for $M(i-1,j)$ and so on. Only the one nucleotide transition in Memo(1,j) is newly calculated for every increment of i.

three sets of ORF's, one set for each frame ($O(F_i)$). Each ORF is then run through the standard BS calculation with the query protein pHMM. The sum of the BS for all ORFs in each frame is then the total BS for that frame ($BS(F_i)$). Finally, the three frame BSs are summed as probabilities to get the windows full BS ($BS(X)$). (Eq. 2.9).

$$NS(F) = \frac{L}{3} * log_2\left(\frac{\frac{L}{3}}{\frac{L}{3}+1}\right) + log_2\left(\frac{1}{\frac{L}{3}+1}\right) \qquad NS(X) = log_2(3 * 2^{NS(F)}) \tag{2.8}$$

$$BS(F_i) = \sum_{j=1}^{O^{F_i}} BS(O_j^{F_i}) \qquad BS(X) = log_2\left(\sum_{i=1}^{3} 2^{BS(F_i)}\right) \tag{2.9}$$

After adjusting the FA Forward score with the DNA window NS and BS, a FA Forward p-value is calculated. FA Forward scores are assumed to follow the same fat tailed distribution as standard Forward scores, but with a different value of μ and τ from the pHMM file. Simulations to validate this assumption are still needed (see Discussion). A target-query pair pass the FA Forward filter if they generate a p-value bellow 1e-5.

At this point FATHMM performs comparison between the FA Forward p-value of the DNA window and the standard Forward p-value(s) of any Viterbi filter-passing ORFs that were used to create that window (more than one may exist if overlapping windows were merged). Depending on the results of this comparison, FATHMM will determine whether possible homology between the target and query is better explained by using a the protein pHMM of the FA codon model. The pipeline then follows one of two parallel tracks, one for standard translated alignment and the other for FA translated alignment.

There are four possible scenarios under which this comparison can take place: (1) both p-values fail to pass, (2) only the FA p-value passes, (3) only one or more of the ORF's standard p-values pass or (4) both the FA and standard p-values pass. The first scenario is the simplest; if neither p-value passes, then the target-query pair is discarded as non-homologous. Under the second scenario, only the FA p-value is suggestive of homology, and the DNA window and FA codon model are passed to

the Domain Definition state along the FA branch. Under the third scenario, the potential homology is limited to the ORFs with passing Forward p-values, and only they and the protein PHMM are passed the the Domain Definition state along the standard (non-FA) branch. The forth and final scenario is the most complicated. One approach might be to run both branches and wait to select the one with the highest e-value just before outputting results to the user, but this would result in an unnecessary increase in both runtime and memory usage. Instead, FATHMM seeks to select the alternative with the greatest support: the scores of all standard-ORF Forward scores in the active DNA window are summed, and used to create a new non-FA Forward p-value. If this is lower than the FA Forward p-value, then the standard branch is used; otherwise the FA branch is used.

2.3 Domain Definition

The algorithms described in this section were originally designed for protein-to-protein alignment. Their goal was to identify the boundaries of one or more copies of a protein domain in a full length protein target (a protein domain is a contiguous run of amino acids that folds independently from the rest of the protein). In nhmmer [22], the process of "Domain Definition" is repurposed to find the boundaries of of homologous regions within the larger DNA window. By isolating these target sub-sequences where the support for homology is high (which I will refer to as domains), the Domain Definition stage supports more accurate alignments and scores while reducing the memory usage and run time needed to produce them.

FATHMM uses the original protein-to-protein Domain Definition algorithms for target query pairs that are passed to the non-FA branch. For pairs that passed to the FA-branch FATHMM uses Domain Definition in the same way as nhmmer, but with modified FA algorithms. The first of these algorithms is Backward. As the name suggests, Backward is essentially just Forward in reverse, summing the probabilities of all possible alignments starting at the last residue in the target and the T state in the model. Depending on which pipeline branch was selected by Forward p-value comparison, the target-query pair will either be run through standard Backward (Fig. 2.12) or a memoized FA Backward (Fig. 2.13).

With complete Forward and Backward matrices, it is possible to compute the posterior probability that each cell in the matrix is part of a correct alignment path through the DP matrix. Each cell in the Forward matrix holds the sum of the probabilities of all paths starting from the S state and the first target residue and ending at that cell, while the corresponding cell in the Backward matrix holds the sum of the probabilities of all paths beginning at that cell and ending at the T state and the last target residue. Multiplying the probabilities in the same cell from Forward and Backward matrices and normalizing by the full matrix probability (i.e. the Forward alignment

Backward

/* Special conditions for 0^{th}, L^{th} and M^{th} indices are omitted for simplicity */

for *i in* {*L-1..0*} **do** // For each residue in the target

  for *j in* {*M..1*} **do** // For each position in the core model

    $B(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{B(i)} \\ 2^{M(i+1,j)+T(B,M)+E(M_J(x_{i+1}))} \end{array} \right\} \right)$; // B state

  end

  $J(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{B(i)+T(J,B)} \\ 2^{J(i+1)+T(J,J)} \end{array} \right\} \right)$; // J state

  $C(i) \leftarrow C(i+1) + T(C,C)$; // C state

  $E(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{J(i)+T(E,J)} \\ 2^{C(i)+T(E,C)} \end{array} \right\} \right)$; // E state

  $N(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} B(i) + T(N,B) \; N(i+1) + T(N,N) \end{array} \right\} \right)$; // N state

  for *j in* {*M..1*} **do** // For each position in the core model

    $M(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{E(i)+T(M,E)} \\ 2^{M(i+1,j+1)+T(M_j,M_{j+1})+E(M_{j+1}(x_{i+1}))} \\ 2^{I(i+1,j)+T(M_j,I_j)} \\ 2^{D(i,j+1)+T(M_j,D_{j+1})} \end{array} \right\} \right)$; // M state

    $I(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{M(i+1,j+1)+T(I_j,M_{j+1})+E(M_{j+1}(x_{i+1}))} \\ 2^{I(i+1,j)+T(I_j,I_j)} \end{array} \right\} \right)$; // I state

    $D(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{E(i)+T(D,E)} \\ 2^{M(i+1,j+1)+T(D_j,M_{j+1})+E(M_{j+1}(x_{i+1}))} \\ 2^{D(i,j+1)+T(D_j,D_{j+1})} \end{array} \right\} \right)$; // D state

  end

end

Figure 2.12: The Backward DP algorithm begins at the ends of the target and query, pulling summed probabilities from the bottom right to the top left of the DP matrix. Another important difference with Forward is that emissions scores for target residue x_i are not included in the match states at row i but pulled into the sum at row $i+1$ by the cells transitioning from the emitting match state.

Memoized Frameshift Aware Backward

/* Special conditions for 0^{th}, L-$(L-5)^{th}$ and M^{th} indices are omitted for simplicity */

for *i in* {*L-1..0*} **do** // For each residue in the target

  for *j in* {*M..1*} **do** // For each position in the core model

    $Memo(j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{M(i+1,j)+E(M_J(x_{i+1}))} \\ 2^{M(i+2,j)+E(M_J(x_{i+1},x_{i+2}))} \\ 2^{M(i+3,j)+E(M_J(x_{i+1},x_{i+2},x_{i+3}))} \\ 2^{M(i+4,j)+E(M_J(x_{i+1},x_{i+2},x_{i+3},x_{i+4}))} \\ 2^{M(i+5,j)+E(M_J(x_{i+1},x_{i+2},x_{i+3},x_{i+4},x_{i+5}))} \end{array} \right\} \right)$; /* Memoization of emissions used by B, M, I and D states */

    $B(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{B(i)} \\ 2^{Memo(j)+T(B,M)} \end{array} \right\} \right)$; // B state

  end

  $J(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{B(i)+T(J,B)} \\ 2^{J(i+3)+T(J,J)} \end{array} \right\} \right)$; // J state

  $C(i) \leftarrow C(i+3) + T(C,C)$; // C state

  $E(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{J(i)+T(E,J)} \\ 2^{C(i)+T(E,C)} \end{array} \right\} \right)$; // E state

  $N(i) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{B(i)+T(N,B)}) \\ 2^{N(i+3)+T(N,N)} \end{array} \right\} \right)$; // N state

  for *j in* {*M..1*} **do** // For each position in the core model

    $M(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{E(i)+T(M,E)} \\ 2^{Memo(j)+T(M_j,M_{j+1})} \\ 2^{I(i+3,j)+T(M_j,I_j)} \\ 2^{D(i,j+1)+T(M_j,D_{j+1})} \end{array} \right\} \right)$; // M state

    $I(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{Memo(j)+T(I_j,M_{j+1})} \\ 2^{I(i+3,j)+T(I_j,I_j)} \end{array} \right\} \right)$; // I state

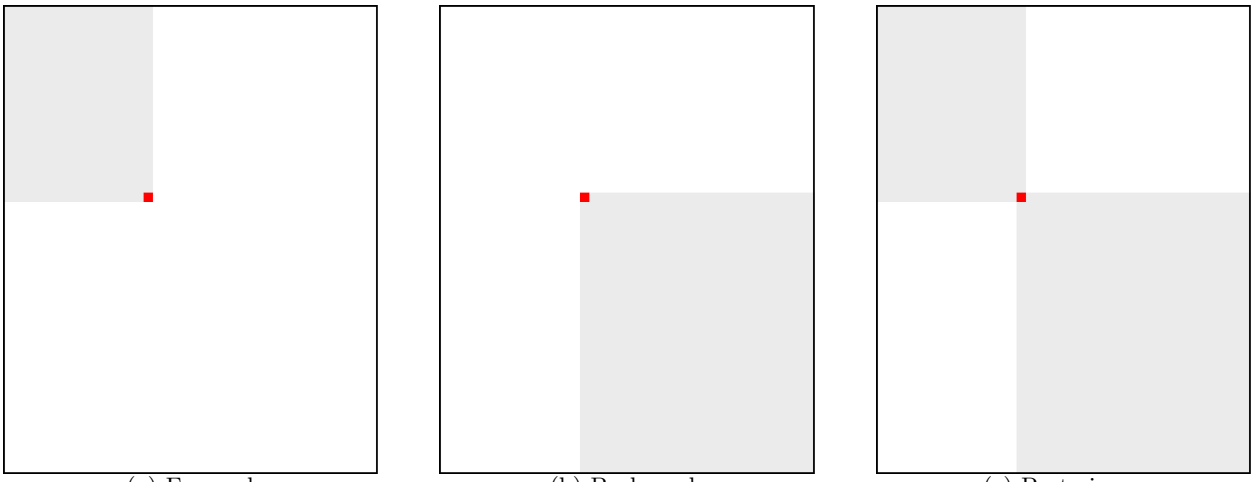
    $D(i,j) \leftarrow log_2 \left(\sum \left\{ \begin{array}{l} 2^{E(i)+T(D,E)} \\ 2^{Memo(j)+T(D_j,M_{j+1})} \\ 2^{D(i,j+1)+T(D_j,D_{j+1})} \end{array} \right\} \right)$; // D state

  end

end

Figure 2.13: In a FA Forward matrix the cell $M(i,j)$ contains the emissions scores for all codons and pseudo-codons emitted by the j^{th} match state and ending with the target nucleotide x_i. In a FA backward matrix cell $M(i,j)$ contains the emissions scores for all codons and pseudo-codons emitted by the $j+1^{th}$ match state and starting with the target nucleotide x_{i+1}. This distinction effects the way in which FA Backward can be memoized, and prevents multiple rows from using the same memoized values. Still there are reusable values which can reduce the total number of operations from for the M, I, D and B states at each i,j pair from 93 in a naive implementation to 48 for the implementation shown above.

score) produces a total probability of all alignments going through that cell (Fig. 2.14).



(a) Forward (b) Backward (c) Posteriors

Figure 2.14: In a filed Forward matrix (a) each i,j cell (such as the one shown in red) contains the probability of all possible alignments between the target sub-sequence $1..i$ and the query sub-model $1..j$ (shown at the grey area in (a)). In a filed Backward matrix (b) each i,j cell contains the probability of all possible alignments between the target sub-sequence $i..L$ and the query sub-model $j..M$ (shown at the grey area in (b)). By multiplying these probabilities in a posterior matrix (c), each i,j cell will contain the probability of any possible alignment between the target and the query that contains that i,j cell.

Only special state posteriors are needed for finding the boundaries of a domain, allowing Forward and Backward to keep a fixed number of rows for the core model's portion of their matrices. Since the calculations in the standard versions of Forward and Backward never need values more than one index away from the current row, the $L*(3M+5)$ matrices are replaced with $(L*5)+(2*3M)$ matrices. Two rows are used (and reused) to calculate values for the core model matrix, with one tracking the current target index (i), while the other tracks the previous target index ($i-1$ for Froward and $i+1$ for Backward). Meanwhile, all computed special state values are retained. The pseudocode in Figure 2.15 shows how, for every residue in the target (i), posterior probabilities are calculated for the B and E states (BTotal and ETotal respectively), and how a single heuristic posterior is calculated for all states in the core model (MTotal).

The FA versions of Forward and Backward are also run with reduced matrices, but with six core model rows instead of two. Thus, the FA branch must also rely on the probabilities in the special states to compute heuristic posteriors (Fig. 2.16). To account for the existence of multiple frames,

Non-FA Heuristic Posteriors for Domain Definition

/* Special conditions for 0^{th} index are omitted for simplicity */

$TotalScore \leftarrow fwd[T]$; // Total score taken from T state of Forward matrix

for *i in {1..L}* **do** // For each residue in the target

  $BTotal[i] \leftarrow 2^{(fwd[B(i-1)]+bck[B(i-1)]-TotalScore)}$; // Probability of entering the core model

  $ETotal[i] \leftarrow 2^{(fwd[E(i)]+bck[E(i)]-TotalScore)}$; // Probability of exiting the core model

  $NJCTotal \leftarrow \sum \left\{ \begin{array}{l} 2^{(fwd[N(i)]+bck[N(i)]-TotalScore)} \\ 2^{(fwd[J(i-1)]+bck[J(i)]+T(J,J)-TotalScore)} \\ 2^{(fwd[C(i-1)]+bck[C(i)]+T(C,C)-TotalScore)} \end{array} \right\}$; /* Posterior probability for x_i being emitted by a special state */

  $MTotal \leftarrow 1.0 - NJCTotal$; // Estimated core model probability

end

Figure 2.15: BTotal and ETotal hold the posterior probabilities of the B and E states which transition in and out of the core model. MTotal holds the estimated probability of residues being emitting by the core model. This estimate is attained by summing the probabilities of a residue being emitted by the N, C or J state and then subtracting that sum from one.

FA Heuristic Posteriors for Domain Definition

/* Special conditions for 0–3^{rd} and $L-1$–L^{th} indices are omitted for simplicity */

$TotalScore \leftarrow fwd[T]$; // Total score taken from T state of the Forward matrix

for *i in {1..L}* **do** // For each residue in the target

  $BTotal[i] \leftarrow \sum \left\{ \begin{array}{l} 2^{(fwd[B(i-3)]+bck[B(i-3)]-TotalScore)} \\ 2^{(fwd[B(i-2)]+bck[B(i-2)]-TotalScore)} \\ 2^{(fwd[B(i-1)]+bck[B(i-1)]-TotalScore)} \end{array} \right\}$; /* Probability of entering the core model */

  $BTotal[i] \leftarrow \sum \left\{ \begin{array}{l} 2^{(fwd[E(i)]+bck[E(i)]-TotalScore)} \\ 2^{(fwd[E(i+1)]+bck[E(i+1)]-TotalScore)} \\ 2^{(fwd[E(i+2)]+bck[E(i+2)]-TotalScore)} \end{array} \right\}$; /* Probability of exiting the core model */

  $NJCTotal \leftarrow -\infty$; // $2^{-\infty} = 0$

  for *c in {i..i+2}* **do** // For each codon containing x_i

    $NJCTotal \leftarrow \sum \left\{ \begin{array}{l} 2^{NJCTotal} 2^{(fwd[N(c)]+bck[N(c)]-TotalScore)} \\ 2^{(fwd[J(c-3)]+bck[J(c)]+T(J,J)-TotalScore)} \\ 2^{(fwd[C(c-3)]+bck[C(c)]+T(C,C)-TotalScore)} \end{array} \right\}$; /* Sum N, J and C state posteriors for codon $x_{c-2}x_{c-1}x_c$ */

  end

  $MTotal \leftarrow 1.0 - NJCTotal$; // Estimated core model probability

end

Figure 2.16: In the FA DP matrices the probabilities of the special states emitting the target residue x_i are spread out over three frames. The heuristic posteriors for each row are therefore summed over all codons containing x_i.

the FA posteriors for the i^{th} row sums the probabilities for all codons that contain the nucleotide x_i. Only one frame at a time will be the true translation of the domain, and therefore hold the majority of probability. By summing all three frames in the special states, we ensure that every index of the heuristic posteriors contains the correct frame's probability, even if the core model has

shifted the domain from one frame to another by emitting a pseudo-codon.

Once the heuristic posteriors have been calculated, both branches of FATHMM use the same thresholds to find the probable start and end points of a domain (Fig. 2.17). The start is identified based on accumulated probability for use of the B state, and the end based on accumulated probability for use of the E state. The sub-sequence of the target that falls between these start and end points delineates a "domain", which is then passed on to the Output Generation Stage where it is realigned and rescored before being reported to the user.

Domain Definition

```
   /* Special conditions for 1st and L^th indices are omitted for simplicity        */
 1 InDomain ← FALSE ;                                          // Begin not in a domain
 2 for i in {1..L} do                                      // For each residue in the target
 3 |   if InDomain = FALSE then                                        // If not in a domain
 4 |   |   if MTotal[i] − BTotal[i] < 0.10 then      // If x_i is a good place to state a domain
 5 |   |   |   p ← i ;                          // Make x_i the first residue in the next domain
 6 |   |   end
 7 |   |   if Mtotal[i] > 0.25 then                                     // If x_i is in a domain
 8 |   |   |   InDomain ← TRUE ;                    // There is a domain that includes x_i
 9 |   |   end
10 |   else                                                                  // If in a Domain
11 |   |   if MTotal[i] − ETotal[i] < 0.10 then        // If x_i is a good place to end a domain
12 |   |   |   q ← i ;                          // Make x_i the last residue in the current domain
13 |   |   |   ProcessDomain(p, q) ;                      // Realign the domain from x_p to x_q
14 |   |   |   InDomain ← FALSE ;                            // No longer in the last domain
15 |   |   |   p ← i + 1 ;                                // New possible start of next domain
16 |   |   end
17 |   end
18 end
```

Figure 2.17: A Domain is defined as a sub-sequence of the target from x_p to x_q, where at least one residue x_r ($p \leq r \leq q$) has an estimated core model posterior probability greater than the heuristic threshold of 0.25. The Domain begins and ends as close as possible to x_r such that the difference between the probability of a residue or codon being emitted by the core model (MTotal), and the probability of of that residue or codon being the first or last in the domain (BTotal and ETotal), is less than the heuristic threshold of 0.1.

2.4 Output Generation

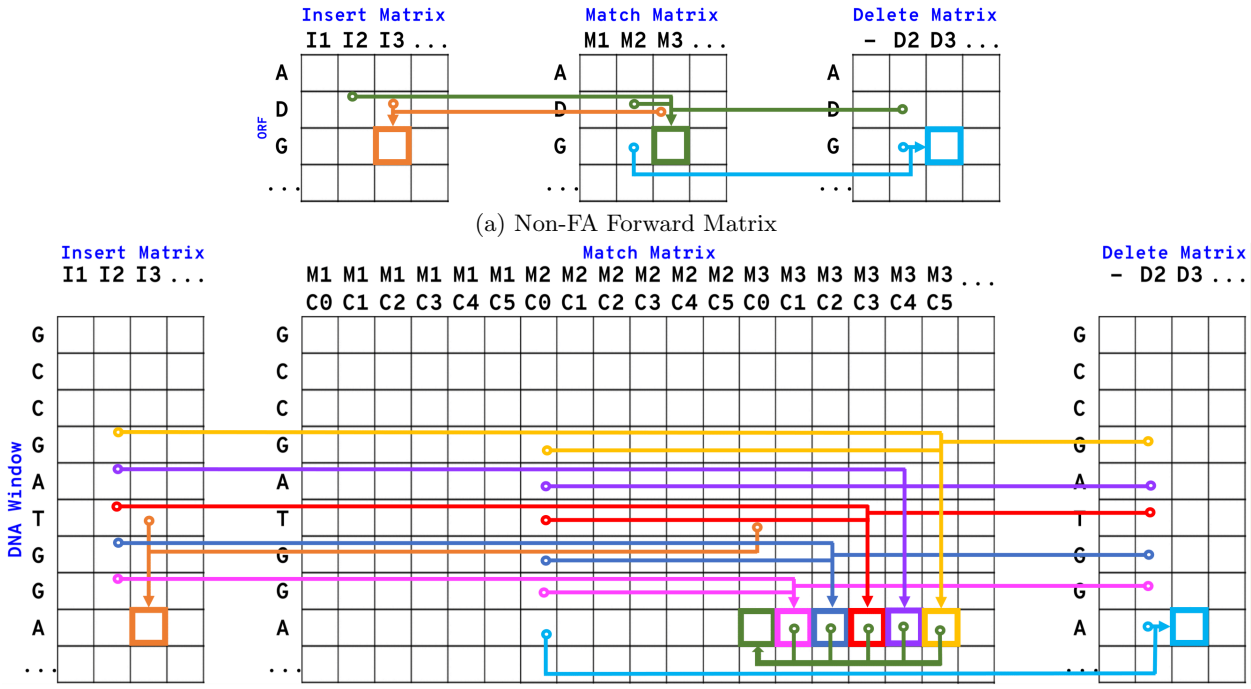
At the end of the Domain Definition stage we have the coordinates of a target region ("domain") with a high probability of homology to the query. At the end of the Output Generation stage we will have an probabilisticly optimal alignment between the target domain and query model, a score for that alignment, and an e-value measuring the significance of that score.

In the introduction I briefly touched on how a maximum scoring alignment can be recovered from a Viterbi matrix be retracing the max functions, which is the method used by many substitution matrix based alignment tools. When using pHMMs we have the advantage of fully probabilistic scores which allow us to calculate posterior probabilities. Alignments recovered from posterior probabilities matrices have been shown to be superior to those recover from Viterbi matrices [23]. The HMMER alignment tools, including FATHMM, build alignments from posterior probabilities in three steps.

The first step is to rerun Forward and Backward on the target domain and query model, but with one less special state and with a full set of L rows for the core model. Since the domain as already been identified as an uninterrupted homologous region, the model is altered to eliminate the J state, updating the N and C state transition scores so that they each have an expected emitted sequence length of $\frac{L}{2}$. For the non-FA branch of the FATHMM pipeline Forward and Backward now use are $L*(3M+4)$ sized matrices, but in the FA branch Forward uses a $L*(8M+4)$ matrix. The extra columns all belong to the match states so that each $M(i,j)$ has six cells, indexed with a third dimension c to become $M(i,j,c)$ Each cell $M(i,j,c)$, $1 \leq c \leq 5$, holds the Forward score for the alignment of a length c (pseudo-)codon ending with the nucleotide x_i to the j^{th} match state. The cell $M(i,j,0)$ holds the summed probability of the other five (Fig. 2.18).

The second step is to calculate the posterior probability of every cell in the Forward and Backward matrices. As with the heuristic posteriors used in domain definition this involves multiplying the probability in each cell of the Forward matrix by the probability in that same cell of the Backward matrix and then diving by the total probability from the end of the Forward matrix. An additional step is then taken to normalize the probabilities of the emitting states (N, C, M and I) across each row. In a standard posterior probability matrix this causes the sum of all those columns for row i to equal one with each cell holding the probability that the residue corresponding to that row was emitted by the state corresponding to that column (Fig. 2.19).

HMMER uses these posteriors not only for alignment recovery but also for a final bias score correction. For FATHMM these two purposes cannot be achieved from the same set of posterior



(a) Non-FA Forward Matrix

(a) FA Forward Matrix

Figure 2.18: The two toy DP matrix examples above demonstrate how the Frameshifts Aware approach differs from a standard DP algorithm. The matrices on the top show the transitions in a standard Forward algorithm where the target and the query are both amino acids or both nucleotides and the match and insert state only emit a single residue from the target. In the Frameshift Aware Forward matrix the query model emits of codons and pseudo-codons while the target is made of nucleotides. Each insert states emission includes three target nucleotides, and the match state can emit anywhere from one to five target nucleotides. When FA Forward is run on a target domain each match state emission length is stores in it's own cell (C1-C5). The sum of all five math emissions is also stored in sixth cell (C0). When transitioning from a match state it is this summed match state score which is used, while the separate scores are used in alignment recovery.

probabilities. Thus the FA-branch must create two posterior probability matrices, one for bias correction that has the standard one column per state, making it $L * (3M + 4)$, and one for alignment recovery that uses the same 6 column per match state as the FA Forward, making it $L * (8M + 4)$.

For bias correction, posteriors are calculated and normalized across each row in the same way as in the non-FA branch. For alignment recovery this would normalization make it hard to distinguish which rows are in the correct frame, especially around any frameshifts, making recovery of the true alignment difficult if not impossible. Instead each row is normalized by the summed probability from all codons and pseudo-codons containing x_i. With this approach, each cell in the alignment

Non-FA Posterior Probabilities

```
   /* Special conditions for 0^st index are omitted for simplicity                    */
 1 TotalScore ← fwd[T];
 2 for i in {1..L} do                                          // For each residue in the target
   |
 3 |  D ← 0 ;                                                  /* Scaling factor used to normalize
   |                                                           across emissions of x_i */
 4 |  for j in {1..M} do                                      // For each position in the core model
 5 |  |  PP[M(i,j)] ← 2^(fwd[M(i,j)]+bck[M(i,j)]−TotalScore) ;          // M state posteriors
 6 |  |  PP[I(i,j)] ← 2^(fwd[I(i,j)]+bck[I(i,j)]−TotalScore) ;          // I state posteriors
   |  |                                                       /* Add M and I state posteriors for the
 7 |  |  D ← D + PP[M(i,j)] + PP[I(i,j)] ;                   emission of x_i to the scaling factor */
   |
 8 |  end
 9 |  PP[N_i] ← 2^(fwd[N(i)]+bck[N(i)]−TotalScore) ;                  // N state posteriors
10 |  PP[C_i] ← 2^(fwd[C(i−1)]+T(C,C)+bck[C(i)]−TotalScore) ;         // C state posteriors
11 |  D ← D + PP[N(i)] + PP[C(i)] ;       // Add N and C state posterior to scaling factor
12 |  for jin{1..M} do                                        // For each position in the core model
13 |  |  PP[M(i,j)] ← PP[M(i,j)] ÷ D ;                              // Normalize M state posteriors
14 |  |  PP[I(i,j)] ← PP[I(i,j)] ÷ D ;                              // Normalize I state posteriors
15 |  end
16 |  PP[N(i)] ← PP[N(i)] ÷ D ;                                    // Normalize N state posteriors
17 |  PP[C(i0] ← PP[C(i)] ÷ D ;                                    // Normalize C state posteriors
18 end
```

Figure 2.19: Posterior probabilities are calculated for each cell in the Forward and Backward matrices and stored in a third matrix (PP). For each row in the matrix a scaling factor (D) sums the posteriors of the emitting state and then uses it to normalize those potteries. This allows the non-FA branch to find which state had the maximum probability of emitting each target domain amino acid. The non-emitting D, B, and E state cells of the PP matrix are all set to zero and not shown in the pseudocode.)

recovery FA posterior matrix holds the probability that the residue corresponding to that row was emitted by the state corresponding to that column *as the last nucleotide in the corresponding (pseudo-)codon* (Fig. 2.20).

Once the posteriors have been calculated and normalized, the third step in alignment recovery seeks a Viterbi-like path through the matrix that maximizes the expected accuracy, or sum of posterior probabilities. In the FA branch this is done with special attention to selection of frame in the special states and selection of emissions length in the match states. The optimal path is then stored in a set of arrays, recording the query state, target index and, in the case of the FA-branch, the (pseudo-)codon length of each cell in the path. These arrays are then used to produce human readable alignments. Samples of FATHMM alignments can be seen in the Results section. Every alignment also has a score taken from the domain Forward matrix and then adjusted by

FA Posterior Probabilities for Alignment Recovery

```
   /* Special conditions for 0–2^nd and L−3–L^th indices are omitted for simplicity */
 1 TotalScore ← fwd[T];
   /* Calculate posteriors                                                          */
 2 for i in {1..L} do                                          // For each residue in the target
 3 |   for j in {1..M} do                                     // For each position in the core model
 4 |   |   for c in {0..5} do                                 // For each match state emissions length
 5 |   |   |   PP[M(i,j,c)] ← 2^(fwd[M(i,j,c)]+bck[M(i,j)]−TotalScore) ;   // Posteriors for all M state
   |   |   |    emissions
 6 |   |   end
 7 |   |   PP[I(i,j)] ← 2^(fwd[I(i,j)]+bck[I(i,j)]−TotalScore) ;          // Posteriors for I state
 8 |   end
 9 |   PP[N(i)] ← 2^(fwd[N(i)]+bck[N(i)]−TotalScore) ;                    // Posteriors for N state
10 |   PP[C_i] ← 2^(fwd[C(i−3)]+T(C,C)+bck[C(i)]−TotalScore) ;            // Posteriors for C state
11 end
   /* Normalize posteriors                                                          */
12 for i in {1..L} do                                          // For each residue in the target
13 |   D ← 0 ;                                                 // Scaling factor for normalization
14 |   for j in {1..M} do
15 |   |   D ← D + PP[M(i,j,0)] + PP[I(i,j)] ;                 // M and I state emissions ending in x_i
16 |   |   for c in {2..5} do
17 |   |   |   D ← D + PP[M(i+1,j,c)] ;                       // M state emissions ending in x_i, x_{i+1}
18 |   |   end
19 |   |   D ← D + PP[I(i+1,j)] ;                             // I state emission of x_i, x_{i+1}
20 |   |   for c in {3..5} do
21 |   |   |   D ← D + PP[M(i+2,j,c)] ;                       // M state emissions ending in x_i, x_{i+1}, x_{i+2}
22 |   |   end
23 |   |   D ← D + PP[I(i+2,j)] ;                             // I state emission of x_i, x_{i+1}, x_{i+2}
24 |   |   for c in {4..5} do
25 |   |   |   D ← D + PP[M(i+3,j,c)] ;                       // M state emissions ending in x_i, x_{i+1}, x_{i+2}, xi+3
26 |   |   end
27 |   |   D ← D + PP[M(i+4,j,5)] ;                           // M state emission of x_i, x_{i+1}, x_{i+2}, xi+3, x_{i+4}
28 |   end
29 |   D ← D + PP[N(i,j)] + PP[C(i,j)] ;                      // N and C state emission of x_{i−2}, x_{i−1}, x_i
30 |   D ← D + PP[N(i+1,j)] + PP[C(i+1,j)] ;                  // N and C state emission of x_{i−1}, x_i, x_{i+1}
31 |   D ← D + PP[N(i+1,j)] + PP[C(i+1,j)] ;                  // N and C state emission of x_i, x_{i+1}, x_{i+2}
32 |   for j in {1..M} do
33 |   |   for c in {0..5} do                                 // For each match state emissions length
34 |   |   |   PP[M(i,j,c)] = PP[M(i,j,c)] ÷ D ;              // Normalize M state posteriors
35 |   |   end
36 |   |   PP[I(i,j)] = PP[I(i,j)] ÷ D ;                      // Normalize I state posteriors
37 |   end
38 |   PP[N(i,j)] = PP[N(i,j)] ÷ D ;                          // Normalize N state posteriors
39 |   PP[C(i,j)] = PP[C(i,j)] ÷ D ;                          // Normalize C state posteriors
40 end
```

Figure 2.20: In order to recover FA alignments from a posterior probability matrix we to do two things differently then in a non-FA posterior matrix. First the matrix must provide five columns per match state in order to keep separate posteriors for the five different (pseudo-)codon lengths as well as a sixth column to keep the sum of the other five. Secondly the denominator used to normalize all posteriors in row i must be the sum of all posteriors that include x_i in the codon or pseudo-codon emitted there. This include The I, N and C state codons at rows $i+1$ and $i+2$ as well as M state codons and pseudo-codons on rows $i+1$, $i+2$, $i+4$ and $i+5$.

null and bias scores. The null score introduces the background model transition probabilities (the denominator of the log odds ratio), and is calculated as during the Filtering stage (e.q. 2.3 for the non-FA branch and e.q. 2.8 for the FA branch), but with L now being the length of the domain. The bias adjustment accounts for the extent to which the background model emissions probabilities do not reflect the bias in the composition of the target and query. The new null score is calculated with L equaling the length of the domain and the new bias scores are calculated using the a posterior probability (PP) matrices.

For hmmsearcht and the non-FA branch of FATHMM this is done with the same posterior probability (PP) matrix as the alignment recovery, but the FA branch uses a separate PP matrix built just for bias scoring. The PP matrix is used to build a set of bias emission scores, one for each amino acid. This is done by first finding, for every emitting state in the query model (M, I, N and C) the average PP over all L of that state's cells in the PP matrix. Remember that the PP matrices used to find the bias score have been normalized so that the M, I, N, and C state PP in a single row of the matrix will sum to one. This means that the average PP of a column will give us the posterior probability that the state was used in the alignment.

Each match state's average PP is then used to weight it's emissions scores. These weighted scores are then summed across all match states, and across the average PP of the N, C and I states, producing an expected frequency for each amino acid or (pseudo-)codon (Eq. 2.10). The bias score for the domain X' is the sum of the bias emissions scores, $BS(x_i)$ for each amino acid or (pseudo-)codon emitted by one of the states. For the non-FA branch the target amino acids are all emitted by one state or another so this simply means summing across the length to the domain (Eq. 2.11). For the FA-branch the alignment is used to determine which codons and pseudo-condos are part to the path trough the matrix and sum the bias scores just for those emissions.

$$BS(x_i) = log_2\left(\sum_{j=1}^{M}\left(E(M_j(x_i)) * \overline{PP(M_j)} + \overline{PP(I_j)}\right) + \overline{PP(N)} + \overline{PP(C)}\right) \tag{2.10}$$

$$BS(X') = log_2\left(2^{\sum_{i=1}^{L}(BS(x_i))+\omega} + 1\right) \tag{2.11}$$

By subtracting the new bias score and null score from the Domain's Forward matrix score, a final hit score is achieved for the target query pair, and a p-value is generated using the same distribution as the Forward filter. To this point in the pipeline, p-values have been sufficient for determining if a target query pair warrants further consideration, but now a correction for multiple testing must be done before the hit and it's significance can be reported to the user. For hmmsearcht, each translated ORF is considered to be a distinct test, so that a hit's e-value is simply the product of its p-value and the total number of ORFs. This same procedure is used for FATHMM hits coming from the non-FA branch of the pipeline. However, the use of DNA windows in the FA branch makes the relationship between each test and the number of ORFs less clear. For these hits I instead applied the multiple-testing correction used by HMMER's DNA to DNA alignment tool, nhmmer [22]: the number of unique tests is set as the total number of nucleotides in all target sequences (and reverse complements) divided by the pHMM's MAXL value (the maximum plausible length of sequences emitted from the model). This quotient gives an approximate number of separate windows in all target sequences, which is then be used as the number of independent tests to convert the FATHMM FA pipeline branch p-values into e-values. Regardless of the branch, FATHMM (under default settings) discards any hit with an e-value above 10.

Any hits passing this final significance test are reported to the user. The user can opt to see the alignment for each hit, and to have each hit's score, bias, e-value, and alignment coordinates reported in a tabular format.

CHAPTER 3 RESULTS

3.1 Benchmark results

3.1.1 Constructing a benchmark

In order to assess FATHMM's performance I tested it on both a set of manufactured benchmarks and a real world data set. The benchmarks are based on the Transmark benchmark created by Walt Shands and Travis Wheeler [personal communication], Transmark is a benchmark for assessing the annotation of protein coding DNA (without frameshifts), and can be used to evaluate tools that perform either translated (protein-to-DNA) or DNA-to-DNA alignment. Transmark is itself based on the protein annotation benchmark Profmark that was developed by Sean Eddy for evaluating protein annotation tools [9].

Transmark was built using MSAs from the protein domain families in Pfam version 27 [2]. Each protein sequence in these MSA's was mapped to its encoding genome, enabling creation of a new MSA of the true protein-coding DNA sequences. Any protein sequence that could not be mapped to the original DNA was discarded, resulting in 14724 protein domain families being turned into DNA MSA's. Each family's DNA MSA was then examined to see if it could be split into test and train sets meeting the following criteria: (1) a minimum of 10 sequences in both the test and train sets, (2) a maximum of 60% identity between any sequence in the test set and any sequence in the train set and (3) a maximum of 70% identity between any two sequences in the test seat. A total of 1463 families were found to meet this criteria, with a total of 27,521 test sequences.

To evaluate false discovery risk, decoy protein-coding DNA sequences were produced, and added a as a supplement to the sequences in the test set from above. A set of 50,000 decoy sequences were

generated; each sequence was produced by first selecting a protein sequence at random from the families not used to make the test and train sets, shuffling the amino acids of the protein sequence, then reverse translating the protein into encoding DNA (randomly selecting an appropriate codon for each amino acid). Half of the test and decoy sequences are reverse complemented to test that the tools can annotate sequences on both strands.

To provide a backdrop for annotation of protein-coding DNA in a genome, Transmark generates 10 simulated chromosomes, each 100 Mbp long, from a 15-sate HMM trained on 1000 randomly selected 100 Kbp chunks of sequence from real archaeal, bacterial and eukaryotic genomes. All test and decoy sequences are then embedded into these simulated chromosomes at randomly selected positions, with the start position and length of every embedding being recorded.

After an annotation tool has been run with the 10 chromosomes as the target and the training sequences as the queries, Transmark reports an analysis of true and false positive rates. A true positive is any hit in which a query aligns to an embedded test sequence of the same family on the correct strand and with at least 50% overlap. A false positive is any hit where a query aligns to a decoy or background sequence. Hits in which a query aligns to a test sequence from a different family, or the same family but the wrong strand, are ignored (i.e. not counted as a false match) because we cannot be sure that the tool is wrong (i.e. perhaps the tool in question has identified a previously un-recognized relationship).

3.1.2 Assessing accuracy of annotation - without frameshifts

In Figure 3.1 we see how FATHMM performs on the frameshift-free Transmark benchmark compared with six other DNA annotation tools (see figure legend for tool details). As the results show, translated alignment tools, with or without frameshift awareness, perform better than DNA-to-DNA alignment tools on protein coding DNA. This is not surprising, since alignment at the level of protein sequence can leverage the greater information content of amino acids relative to nucleotides (larger alphabet, with more nuanced relationship between letters of the alphabet). If the target uses codons not seen in the query, nucleotide alignments will penalize these differences even if the

codons translate to the same amino acid or to one which would be positive scoring if using a protein alignment.

It is also clear that the tools using pHMMs outperform the tools doing the same sort of search using pairwise alignments (e.g. FATHMM does better than LAST and tfasty, while nhmmer does better than blastn). The frameshift aware tools show slightly worse performance than the translated tools using the same type of query, (e.g. FATHMM does worse than hmmsearcht and LAST and tfasty do worse than tbalstn), as the allowance for frameshifts slightly increases the early occurrence of false positives for these tools. FATHMM's performance, however, is much closer to that of hmmsearcht than either LAST or tfasty are to tbalstn.

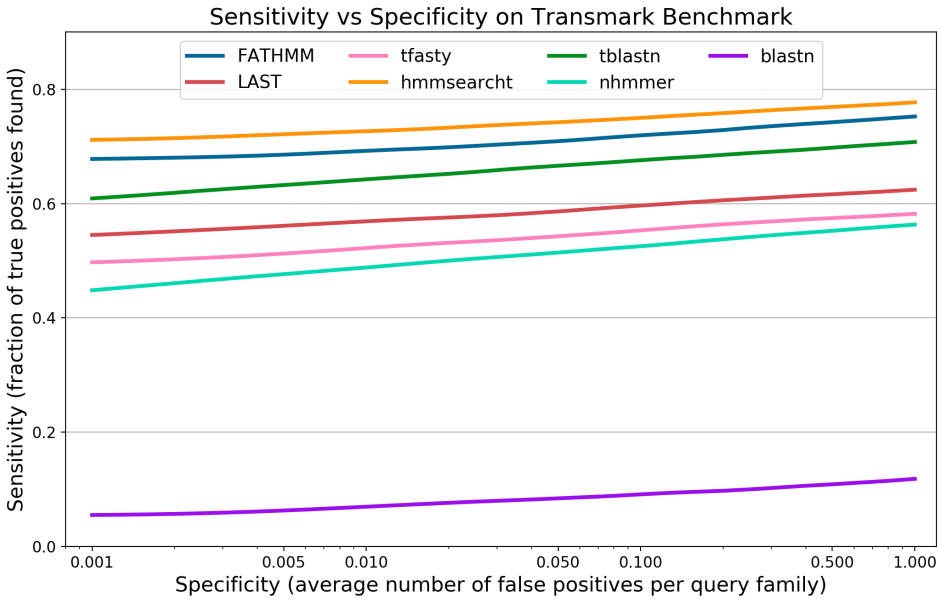


Figure 3.1: Comparison of seven DNA annotation tools on the Transmark benchmark. Two of the tools, nhmmer and balstn, perform DNA to DNA alignments. Two of the tools, hmmsearcht and tblastn, perform translated alignments without accounting for frameshifts. Three of the tools, FATHMM, LAST and tfasty, perform frameshift aware translated alignment. For tools that use pHMMs (FATHMM, hmmsearcht, and nhmmer), each family's set of train sequences are aligned and turned into a query pHMM. For the sequence-to-sequence alignment tools (LAST, tfasty, tblastn and blastn), a separate search is performed for each training sequence and only the best scoring hit for each test sequence is used. The Y-axis measures each tool's sensitivity as the percent of all true positive test sequences that are recovered as a function of the average number of false positive per query sequence family (X-axis).

3.1.3 Assessing accuracy of annotation - with frameshifts

Transmark can only tell us how these tools perform in the absence of frameshifts, so I created four frameshifted versions of Transmark. These benchmarks use the same structure as the original Transmark, except that each test sequence has a non-zero probability of including randomly inserted or deleted nucleotides. These four 'Framemark' benchmarks each have a different frameshift rate, with indels being added to the test sequence in Framemark1 at a rate of 1%, Framemark2 at a rate of 2% , Framemark5 at a rate of 5% and Framemark10 at a rate of 10%. Thus, for every i^{th} nucleotide position in a test sequence in Framemark2, there is a 2% probability that an indel will begin at i. (Note: a 5% indel rate means that a frameshift-inducing indel is expected to appear once every 20 nucleotides, or once every $\sim$7 amino acids, while a 10% indel rate means expected frameshifts every $\sim$3 amino acids - these are exceptionally high rates of frameshift.)

For all Framemark benchmarks, an indel at position i may be either an insertion or deletion with equal probability. For an insertion, a random nucleotide is inserted at position i, and a deletion causes the ith nucleotide to be removed. Each insertion or deletion has a 50% probability of terminating and a 50% probability of extending another nucleotide, repeating until the indel is terminated. Thus, 50% of all indels will be of length 1 nucleotide, 25% will be of length 2, 12.5% will be of length 3, and so on. Since some of these lengths are multiples of three they will not cause frameshifts, slightly lowering the true occurrence of frameshift to about 85% of the stated indel probability.

Figures 3.2, 3.3, 3.4, and 3.5 show the performance of FATHMM and the other tools on the Framemark benchmarks. FATHMM out-performs the other tools on all four of these benchmarks, capturing more true positives with fewer false positives, and even demonstrating modest sensitivity in the extreme case of a 10% indel rate.

For Framemark1 and Framemark2, hmmsearcht (translated pHMMs, with no frameshift model) is the second best performer; inspection of the positive matches shows that hmmsearcht alignments are on average only 48% as long as FATHMM's positive alignments for Framemark1, and only 40% as long for Framemark2. As shown below, FATHMM does not appear to produce inappro-

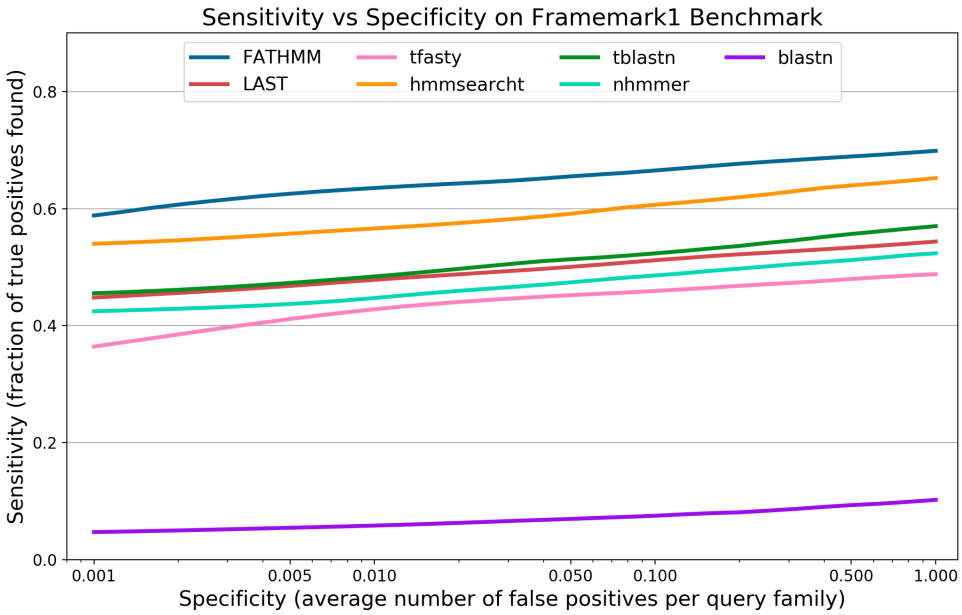


Figure 3.2: Framemark1 has a 1% rate of nucleotide indels.

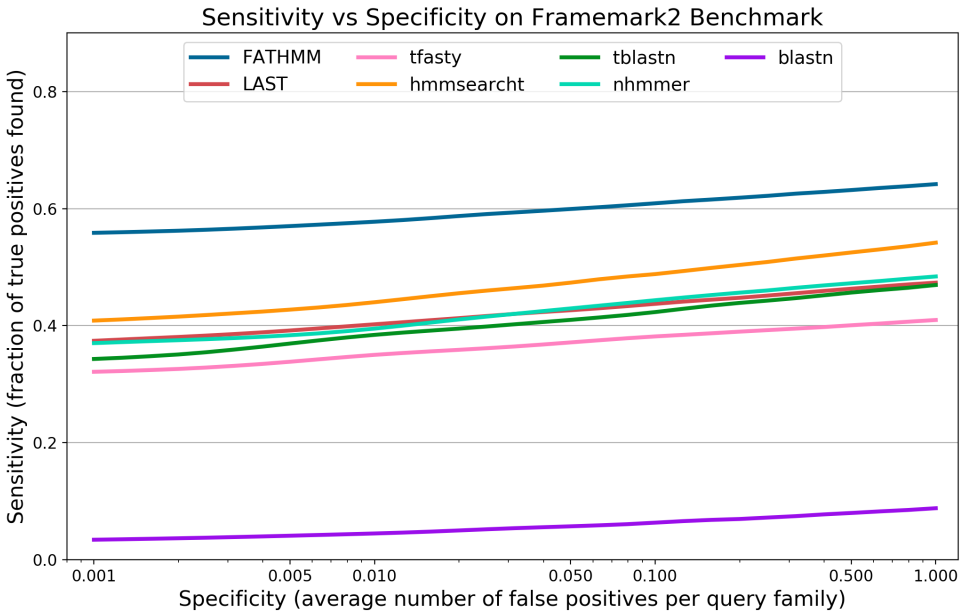


Figure 3.3: Framemark2 has a 2% rate of nucleotide indels.

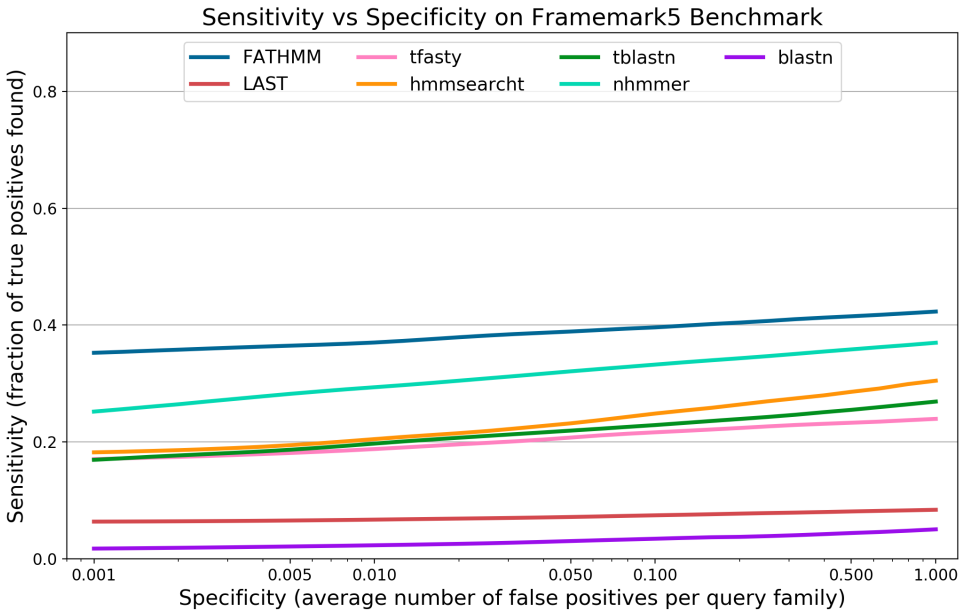


Figure 3.4: Framemark5 has a 5% rate of nucleotide indels.

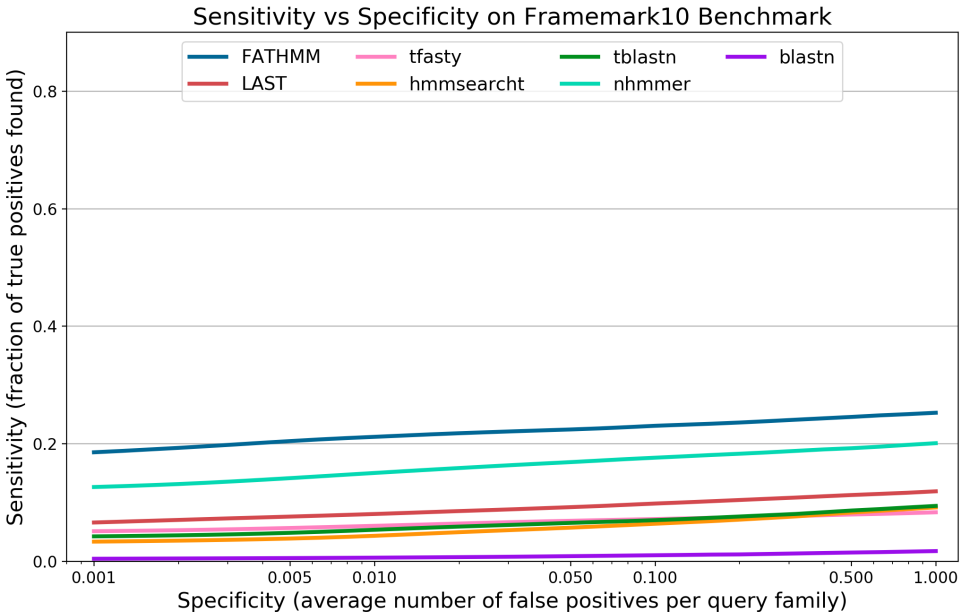


Figure 3.5: Framemark10 has a 10% rate of nucleotide indels.

priately long alignments (so-called overextension), so this difference is attributable to hmmsearcht cutting hits short when encountering a frameshift that FATHMM is capable of accommodating. For Framemark5 and 10, nhmmer performs better than any of the translated search tools except FATHMM. At this level of frameshift nhmmer's ignorance of codons becomes and advantage as it naturally absorbs nucleotide indels. On these benchmarks, FATHMM remains the best-performing tool, as it can account for indels while still benefiting from considering similarity at the level of amino acids. The other frameshift aware tools, LAST and tfasty, do not perform nearly as well as FATHMM. Their reliance on substitution matrices and Viterbi scores prevents them from doing much better than any of the other translated search tools despite attempting to account for frameshifts.

3.1.4 Run time

Because genome annotation involves increasingly large data sets, I also examined the run time of FATHMM relative to other tools. Table 3.1 shows the run times of all seven tools on all five benchmarks when run with 16 threads. All of these programs require O(L*M) worst-case run time, but distinct implementation optimizations and heuristics lead to differences in run times. An important factor is whether L and M refer to the length of the target and query in nucleotides or in amino acids. FATHMM's run time is most easily compared to hmmsearcht and nhmmer since they share the same general architecture. With its current implementation, FATHMM runs an average of $\sim$ 4.5 times slower than hmmsearcht and about the same speed as nhmmer. Considering the greater number of calculations and the fact that the standard versions of Forward and Backward have been implemented using SIMD vector-parallel instructions, these numbers are well within tolerance. Future work to improve run time will be covered in the Discussion.

3.1.5 Overextension

In addition to sensitivity, specificity, and run time, I also used the Transmark and Framemark benchmarks to consider the problems of over-extension, coverage and false frameshifts. Overexten-

Run Times (hours)							
Benchmark	FATHMM	LAST	tfasty	hmmsearcht	tblastn	nhmmer	blastn
Transmark	26.25	7.43	20.96	6.55	5.95	29.33	5.33
Framemark1	29.86	6.35	20.32	4.36	3.88	28.65	11.33
Framemark2	32.36	6.41	17.09	6.05	3.62	26.16	9.02
Framemark5	24.86	6.21	19.39	4.48	4.72	29.76	9.99
Framemark10	28.19	5.36	21.89	4.99	3.25	26.58	14.84
Average	**28.30**	**6.35**	**19.93**	**5.29**	**4.28**	**28.10**	**10.10**

Table 3.1: Run times (in hours) of 7 DNA annotation tools on 5 benchmarks. All tests were run with 16 threads on a system with a 32-core Intel Xeon E5-2630 v3 @ 2.40GHz, and 64 Gb RAM.

sion happens when a tool reports a true positive but extends the alignment beyond the correct start or end points of the target. I had hypothesised that overextension could be a significant problem for FATHMM, as it could shift frames near the start and end of an alignment to pick up a few extra bits of score outside the coordinates of the target true positive. Transmark is the best benchmark to test this hypothesis with because it lacks any implanted frameshifts so that, (1) all frameshifts called by FATHMM are presumably false and (2) frameshifts will not cause ant tool to cut their alignments short.

Figure 3.6 shows occurrences of overextension on Transmark true positive hits. These results make it appear that FATHMM's rate of overexertion (1.1% of true positives with an average length of 9 nucleotides) is not any higher than hmmsearcht's (1.2% of true positives with an average length of 8 nucleotides). However, the fact that the majority of the true positives FATHMM reports for Transmark (82%) come from the non-FA branch could be obscuring the true impact of false frameshifts. To look into this possibility I separated out the relatively small number (3421) of FATHMM hits coming from the FA branch and compared them against the other tools (Fig. 3.7). Here we do see an increase in FATHMM's overextension both in terms of rate (2.9%) and length (average of 10 nucleotides). However, this is still a small percentage of mostly short overextensions

3.1.6 Underextension (completeness of coverage)

Tools that correct too heavily for overextension may encounter the opposing problem of underextension, in which the alignment fails to cover the full length of the true positive. Low coverage can

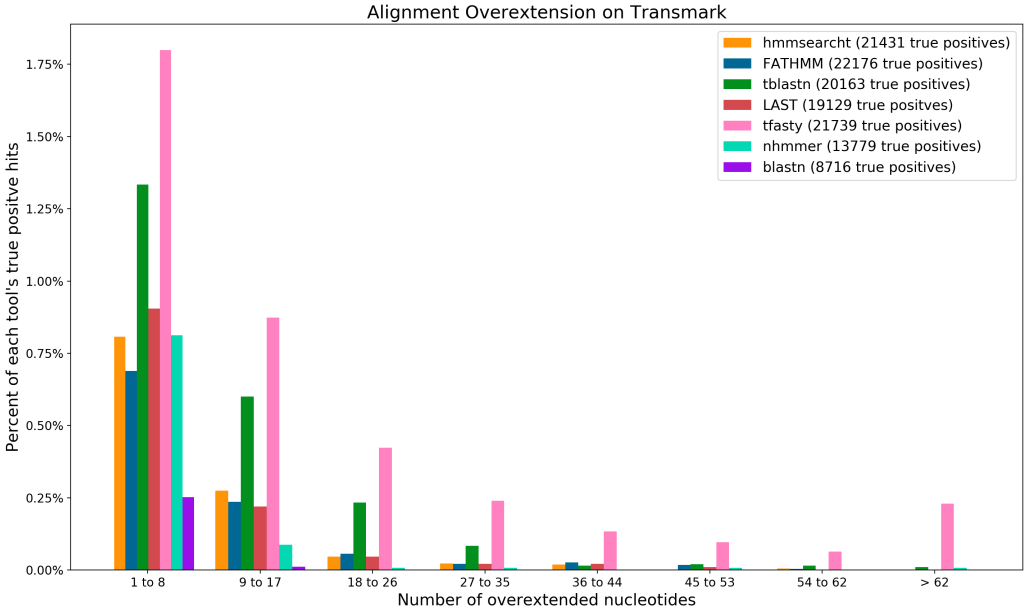


Figure 3.6: The x-axis shows the lengths of the overextensions, in nucleotides, and the height of the bars shows the percent of all true positives found by the tool which were overextended by that particular length. Only true positive hits with e-values less than one were used to make this graph.

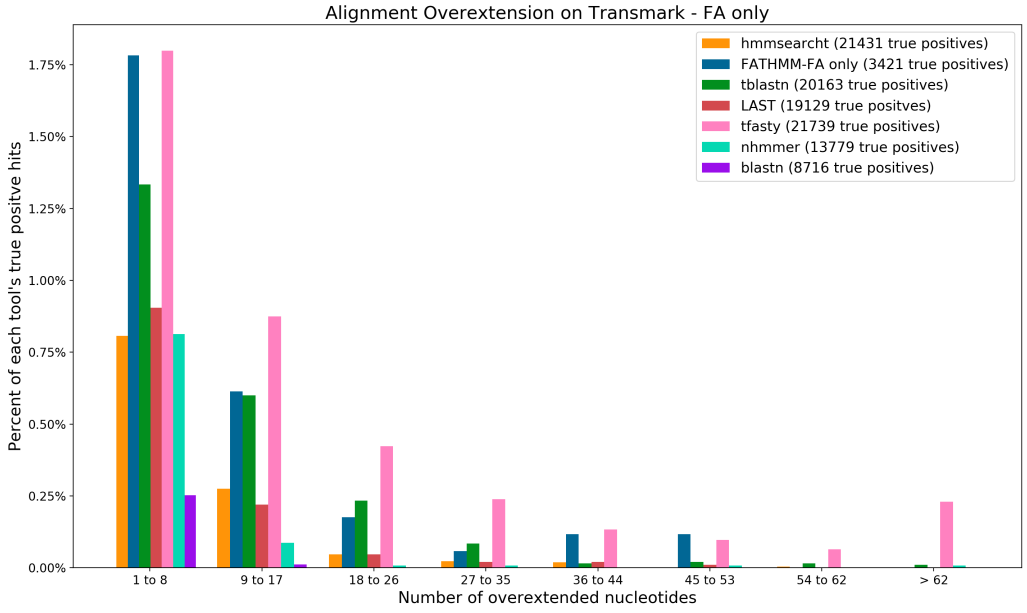


Figure 3.7: The x-axis shows the lengths of the overextensions, in nucleotides, and the height of the bars shows the percent of all true positives found by the tool which were overextended by that particular length. The result for FATHMM only include hits from the FA branch. Only true positive hits with e-values less than one were used to make this graph.

also be caused by low sensitivity if a tool only aligns the sections of a true positive with higher percent identity with the query. We can measure coverage as the percent of the full length of a true positive which is included in the alignment.

Figures 3.8 and 3.9 show coverage for 6 of the 7 tools on Transmark and Framemark1 (blastn was excluded due to poor true positive recovery). This coverage was measured on the set of true positives test sequence which all 6 tools were able to find (i.e. the intersection of positives with E-value $<$ 1, for all 6 tools). In the Transmark results, the lack of alignment-interrupting frameshifts means that frameshift awareness does not bestow any advantage and coverage closely tracks accuracy (Fig. 3.1). However, even at the relatively low 1% frameshift rate of Framemark1, we see that tools able to handle frameshifts (FATHMM, last, and tfasty) produce considerably better coverage of the shared hits, even though two of them (LAST and tfasty) have lower overall accuracy (Fig. 3.2). FATHMM does particularly well in term of coverage on Framemark1, combining high accuracy and robust frameshift awareness.

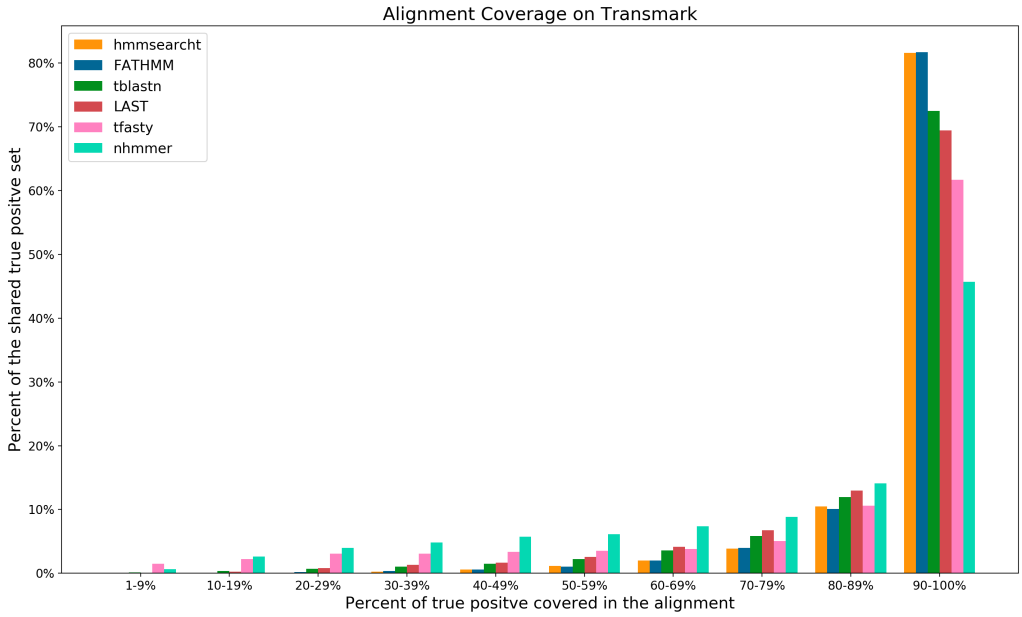


Figure 3.8: A set of 14,680 true positives from Transmark that were found by all 6 tools were used to analyze coverage. Any nucleotide inside the bounds of a true positive that was included in one or more alignments was considered "covered". Coverage is the percent of the total number of nucleotides in the positive target (i.e. the length) that are covered. Coverage percents are binned in 10 coverage ranges (seen on the x-axis) with percent per bin displayed on the y-axis.

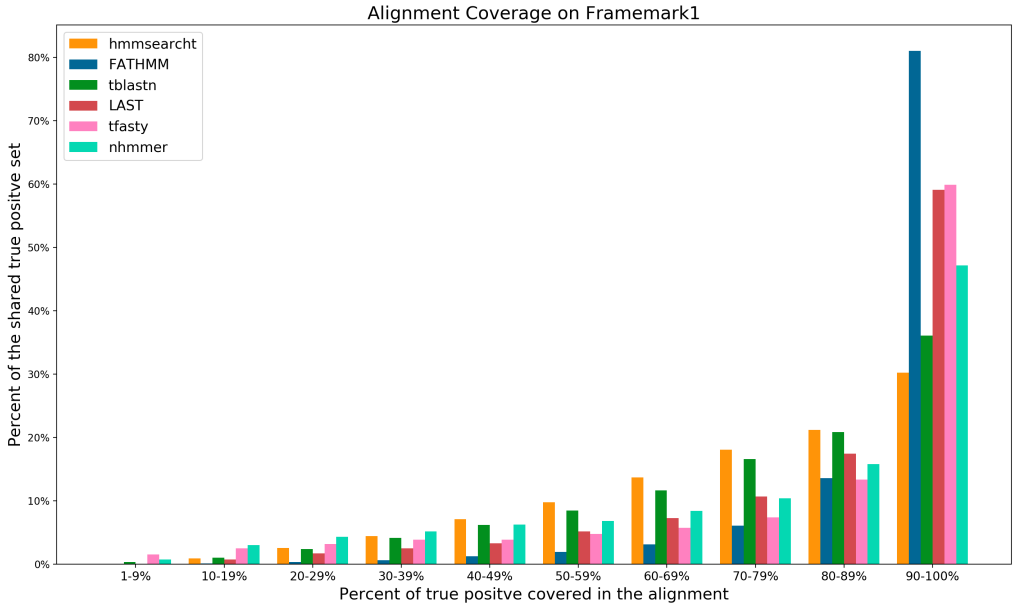


Figure 3.9: A set of 12,709 true positives from Framemark1 that were found by all 6 besides tblastn were used to analyze coverage in the face of frameshifts. Coverage was defined as in the previous figure.

The total coverage for a true positive is computed as the fraction of all letters in the sequence that are covered by any alignment, meaning that the coverage may be broken up over more than one alignment. This is particularly true for the non-frameshift-aware tools on Framemark1. Table 3.2 shows the average number of alignments used to arrive at each true positives coverage. For Transmark, the alignment counts are all essentially one (with the exception of tfasty). As frameshifts are introduced in Framemark1, some tools require multiple alignments to capture segments in different frames, in addition to getting lower coverage. This is particularly true of the non-FA translated tools, hmmsearcht, and tblastn. FATHMM and nhmmer's alignment counts are practically unaffected by the change from Transmark to Framemark1. For nhmmer this is because it is oblivious to codon structure and so is mainly limited by DNA-to-DNA sensitivity. For FATHMM, we see once again that the combination of translated search, pHMM, and frameshift awareness produces superior results.

Number of Alignments Used to Measure Coverage						
Benchmark	FATHMM	LAST	tfasty	hmmsearcht	tblastn	nhmmer
Transmark	1.04	1.06	1.76	1.06	1.11	1.11
Framemark1	1.06	1.29	1.91	1.80	1.73	1.11

Table 3.2: Tools may split coverage between more than one alignment. The table above shows the average number of alignments per true positive used to get the coverage shown in Figures 3.8 and 3.9.

3.1.7 Accuracy in specific frameshift calls

The final analysis sought to learn how often and under what circumstances FATHMM produces false frameshifts (defined as a pseudo-codon in an alignment with no implanted frameshifts). In FATHMM-Transmark hits with e-values less than 1, false frameshifts occur in 5% of all true positives. On average, the FATHMM-Transmark alignments with these false frameshifts contain 2.0 pseudo-codons. This compares to LAST, which produces false frameshifts in 12% of Transmark true positive alignments with an average of 1.7 false frameshift in each. Similarly, tfasty produces false frameshifts in 22% of Transmark true positive alignments with an average of 1.8 false frameshift in each.

To determine the cause of FATHMM's false frameshifts, I looked first at where in the alignments they were occurring. I suspected that false frameshifts might be causing a higher rate of overextension in FA branch alignments, but Figure 3.10 shows that false frameshifts are actually less likely to occur near the ends of alignments and are otherwise evenly distributed. This suggests another reason for false frameshifts occurring nearer to the center of alignments.

Figure 3.11 shows an extreme example of a true positive FATHMM-Transmark alignment with 10 pseudo-codons, in which compensating frameshifts repeatedly return the aligning to the primary frame. Comparing the FATHMM alignment to the hmmsearcht alignment, it is not hard to see why FATHMM called these 10 false frameshifts. The hmmsearcht hit has a score of 96.9 and an e-value of 2.4e-24; it has 126 positive scoring matches, of which 75 are identities (exact matches); it also has 5 amino acid indels. The FATHMM hit has a higher score of 172.4 and an e-value of 1.6e-49. The FATHMM alignment has 162 positive scoring matches, of which 106 are identities, and just 1 amino acid indel. When such significant improvements in alignment quality can be made by calling

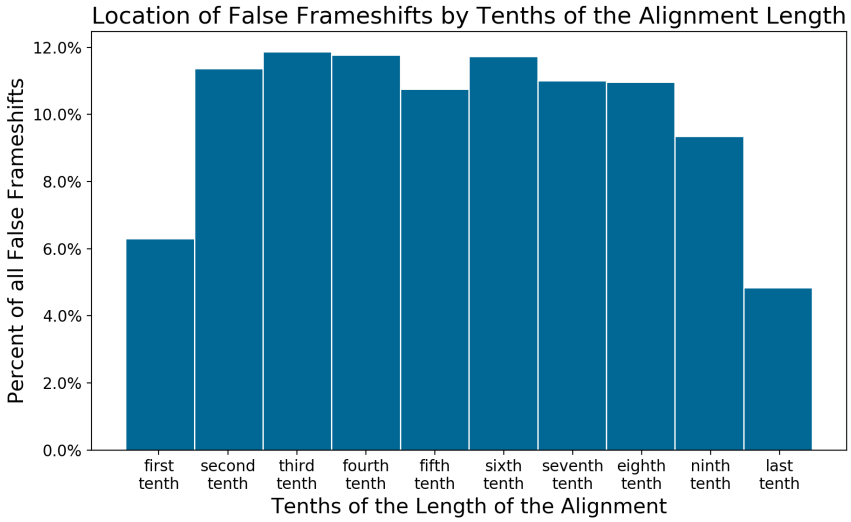


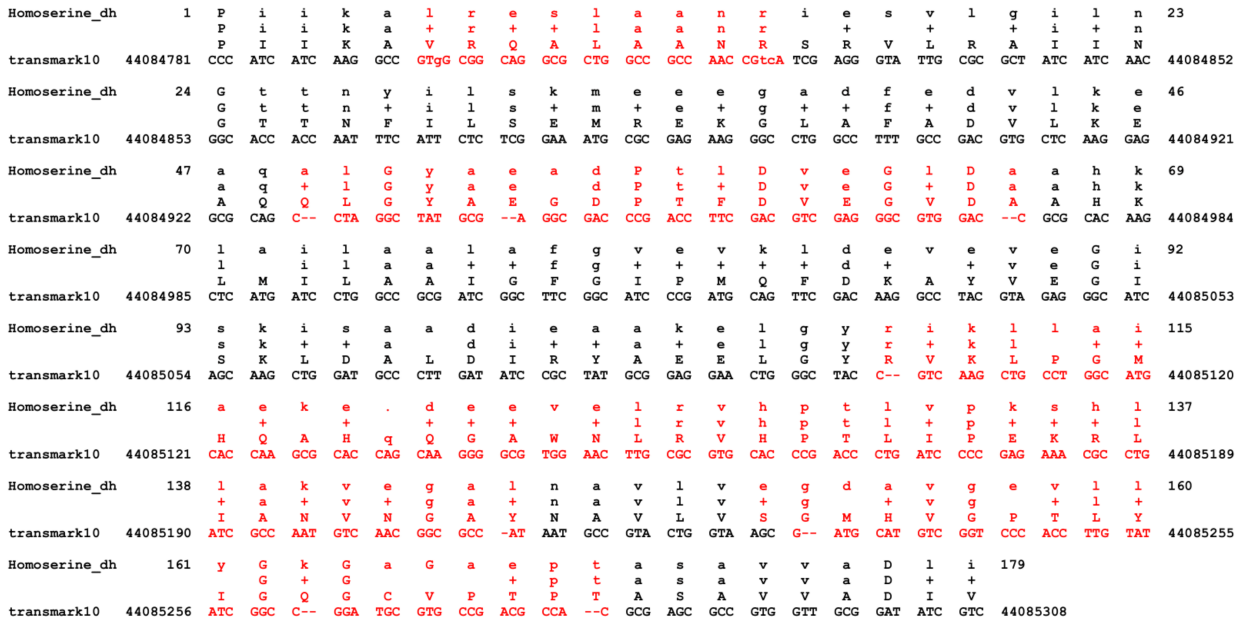
Figure 3.10: Each FATHMM-Transmark alignment that contained at least one false frameshift was cut into ten equal sub-sequences and the number of pseudo-codons in each tenth was counted and plotted above.

false frameshifts they will inevitably occur, but it may be that a lower default frameshift probability (currently 1%) could decrease their frequencies without losing sensitivity on true frameshifts (see Discussion for more details).

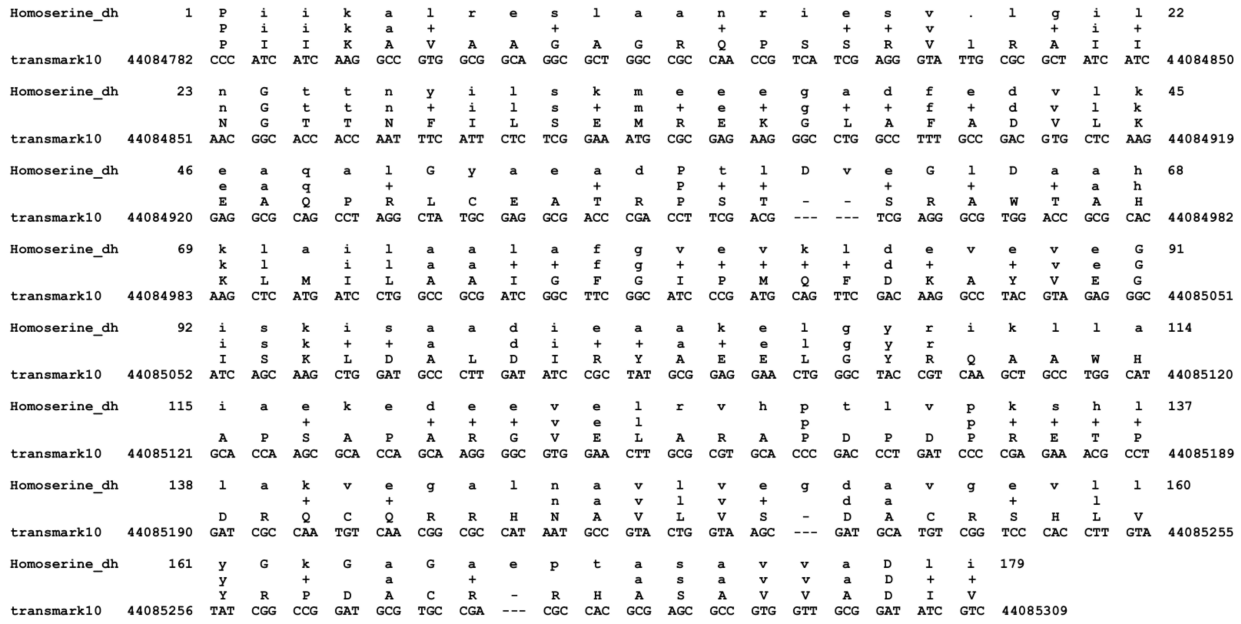
3.2 Application to real sequence data

Analysis on the above simulated benchmark shows that FATHMM appears to provide good performance compared to other tools. However, no manufactured benchmark can perfectly simulate the complexity of real genomic data, so I performed a small-scale analyses on actual frameshifted genes. For this task, I used the genomes of bacterial endosymbionts of a species of cicada.

As is the case with many sap feeding insects, the cicada species *Magicicada tredecim* relies on endosymbionts (in this case *Candidatus Hodgkinia cicadicola*) to provide essential amino acids not found in their food [24] [25]. The long life of *Magicicada tredecim* results in unique evolutionary circumstance for its endosymbionts. These pressures have induced extensive linage splitting, leading to more than 40 separate circular genomes of various sizes, each having their own subset of the original genes. Not all missing genes are entirely lost in these genomes, as some may simply have accumulating function-destroying mutations such as frameshifts (producing inactive remnants of



(a) FATHMM Alignment



(b) hmmsearcht Alignment

Figure 3.11: Alignment of the same target and query from FATHMM (a) and hmmsearcht (b). Each row of the alignments has four lines. The top line is the query consensus sequence and the bottom line is the target DNA, separated into (pseudo-)codons. Above the target DNA is the amino acid translation for those (pseudo-)codons and the line above that indicates positive scoring mismatches with a '+' and exact matches with the amino acid. The FATHMM alignment has 10 false frameshifts and the out of frame regions are shown in red. These alignments demonstrate how false frameshifts can accrue when doing so improves alignment quality.

genes called *pseudogenes*).

Matthew Campbell and John McCutcheon provided me with 43 *Candidatus Hodgkinia cicadicola* genomes found in *Magicicada tredecim* by [personal communication], along with protein multiple sequence alignments from all known *Hodgkinia* genes to use as a query set. The genomes ranged in size from 8821 to 61,247 bp, with an average length of 26,490 bp. I annotated all 43 genomes with both FATHMM and hmmsearcht, and compared the results. FATHMM produced greater coverage than hmmsearcht on all 43 targets (where coverage is the percent of the full length of the genome that is aligned to a query with an e-value less than one). On average, FATHMM increases coverage of the chromosomes by 12% over hmmsearcht. This improvement is from both creating full length alignments where hmmsearcht could find only partial hits, and finding entirely new alignments that hmmsearcht could not identify. Figure 3.12 shows the three genomes with the greatest improvement in coverage from FATHMM. From the top right and going clockwise, these genomes saw a 43%, 28%, and 26% gain in coverage.

In alignments where FATHMM called frameshifts, the average rate at which they were called (measured as the number of pseudo-codons per nucleotide length of the alignment) was 1% (with the range being 0.07% to 3.35%). Figure 3.13 shows a FATHMM alignment of the cobN protein against the largest of the *Hodgkinia* genomes. The alignment has a frameshifts rate of 1.4%, and is 10 times as long as the alignment produced by hmmsearcht for the same query-target pair.

On both benchmarks and real world data FATHMM has been shown to be a powerful tool for the annotation of protein coding DNA in the presence of frameshifts. With the rise in the use of long-read sequencers, which are far more likely to introduce frameshifts than their short-read counterparts, and the continued interest in exploring genomic history through pseudogenes and transposable elements, I believe FATHMM will make an important contribution to the improved annotation of such sequences. Ongoing development (see Discussion), will serve to make it even more valuable going forward.

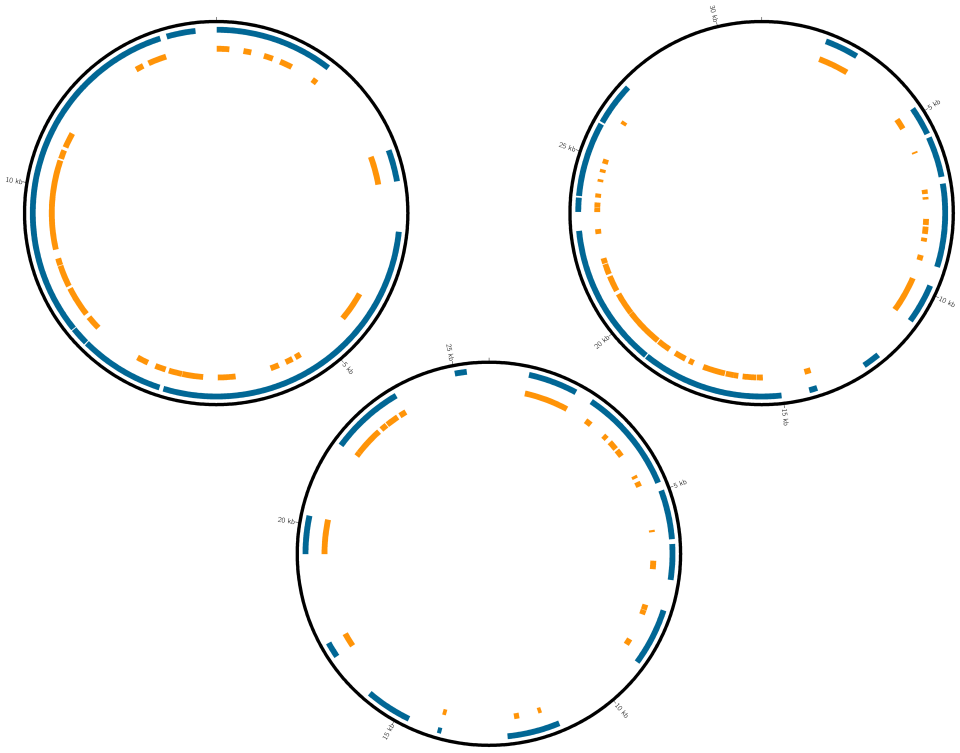


Figure 3.12: Above are four *Candidatus Hodgkinia cicadicola* genomes from *Magicicada tredecim*. The black circle represents the complete circular genome. The blue arcs inside the circle show where FATHMM found hits; the orange arcs show where hmmsearcht found hits.

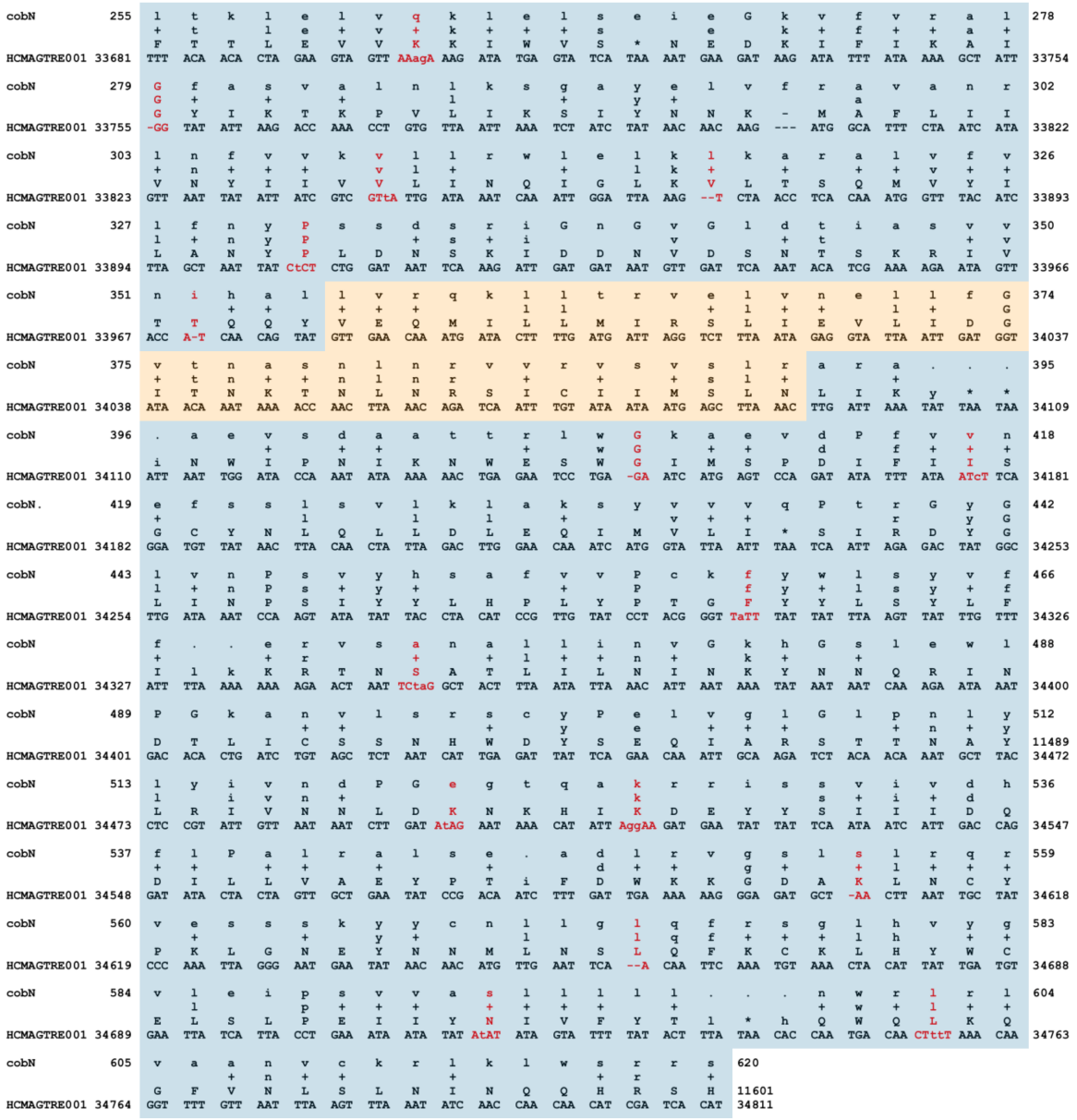


Figure 3.13: Pseudogene Alignment from FATHMM on *Candidatus Hodgkinia cicadicola* genome. This figure contains an alignment of the cobN protein to a pseudogenized region of the largest Hodgkinia genome, with 16 alignment rows serving as word-wrapped segments of a single full-length alignment. Each row of the alignment has four lines. The top line is the query consensus sequence and the bottom line is the target DNA sequence, separated into codons and pseudo-codons. Above the target DNA is the amino acid translation for those (pseudo-)codons. The remaining line, second from the top, indicates positive scoring matches with a '+' and exact matches with the amino acid in question. The area highlighted in orange marks the boundaries of the annotation provided by hmmsearcht. Frameshifts called by FATHMM are shown in red.

CHAPTER 4 DISCUSSION

FATHMM has reached a stage in it's development where it is ready for general use. It has demonstrated a high rate of sensitivity, a low false positive rate, and accuracy in locating frameshifts, all with an acceptable run time. We have also applied it to real data from collaborations, and others are beginning to use the software. Still there are improvements that can be made in all areas.

Run time is an obvious area for potential improvement. One idea is to create SIMD vector parallel implementations of the FA versions of Forward and Backward. Another idea to adapt work done by David Rich [26] to prune the calculation space of Forward and Backward, which results in a typical speedup in that stage of 30-40x. This is achieved by first recognizing that the vast majority of the probability in Forward/Backward matrices exist in a could of cells surrounding the optimal alignment, and then pruning away paths in the matrices that fall outside of that cloud. A possible middle ground is to adopt a pHMM analog to the SMID vectorized block aligner strategy of [27].

It is also likely possible to improve cache performance when accessing the larger range of emissions scores needed for the FA codon model. Currently there is a separate score stored for every one of the 1364 codons and pseudo-codons, resulting in a great deal of redundancy. There are actually only $\sim$60 unique scores, twenty for codons, twenty for length 2 and 4 pseudo-codons, twenty for length 1 and 5 pseudo-codons, plus a few for stop codons. Since the pseudo-codons and stop codons currently map differently at each match state it is not straightforward to devise a universal mapping system that would allow us to only keep one set of those 60 values. Thus, I am stuck with the M*1364 size of the emissions score matrix if I wish to keep using position specific scores for the pseudo-codons.

The logic behind position-specific scoring and the max-function mapping used to create the FA codon model is that it preserves the potential for homology that exists in the pseudo-codons. For instance, if FATHMM maps the pseudo-codon *ac* to the amino acid D (whose codons are *gac* and *gat*) and not to the amino acid E (whose codons are *gaa* and *gag*) it retains the information that the two remaining nucleotides can provide. However, the difference between the emissions probability of D and E for any match state is likely small compared to the frameshift penalty applied to pseudo-codon emissions and it is unclear if the max-functions actually improve accuracy. Future work may involve comparing the current implementation with two simple universal frameshift scores for all pseudo-codons (one for those with a single indel and one for those with two indels).

Another important area for future improvements is the calculation of e-values. As stated in the Methods section, I made an assumption that the FA forward scores followed the same distribution as standard Forward scores when computing p-values both for the Forward filter and for the output e-values. Testing this assumption will require running FATHMM on a very large set of randomly generated targets (with both the MSV and Viterbi filters turned off), or development of an importance sampling approach to reduce the simulation load.

There is also the potential to improve FATHMM's performance on highly frameshifted sequence by addressing another assumption made by the current implementation. In the Methods section I mentioned that FATHMM used translated ORFs as the targets for both the MSV and Viterbi filters, allowing it to use the efficient, SIMD vectorized, standard implementations of these filters already present in the HMMER code base. As the rate of frameshifts in the target DNA sequence increases, the accuracy of these filters will decrease due to mistranslation. It may be that this is inconsequential because any sequence that is too frameshifted to pass MSV and Viterbi is also too frameshifted to pass FA Forward, or it may be a serious limitation of the current implementation. To test this I would need to run FATHMM on Framemark5 and Framemark10 with the MSV and Viterbi filter's turned off to see if they are throwing away a large number of true positives, and possibly implement a Frameshift-aware Viterbi filter.

FATHMM is intended to be useful for identifying instances of protein coding DNA that contains

frameshift inducing indels. These will arise in pseudogenes and in the case of sequencing error. Importantly, in either of these cases, it is also reasonable to expect nucleotide substitutions that may change the encoded amino acid in ways that do not obey expectations represented in emissions scores based on protein MSAs. It will be important to explore potential modifications to the codon emissions scoring scheme, allowing for non-synonymous substitutions.

A final improvement could be made by optimizing the default frameshift penalty and possibly implementing and iterative approach to finding the optimal frameshift penalty for each target-query pair. The current default of 1% probability for each inserted of deleted nucleotide was selected ad hoc and, as of yet, has not bee rigorously tested. To do so, a variety of frameshift penalties would need to by used to run the Transmark and Framemark benchmarks. The results of these tests would tell if one penalty performs best for all benchmarks, in which case I will make that the default, or if it differs depending on the underlying rate of frameshifts. If the results are different for the different benchmarks, as I suspect they will be, I still won't know which penalty is best to use on any particular real world dataset, whose rate of frameshifts is entirely unknown and may differ across the target. In this case the best results may be attained by performing an iterative approach: run a single target-query pair though the Forward filter more than once with a set of increasingly permissive frameshift penalties until the optimal penalty for that pair is identified. The exact details of how, and even whether, such an approach can be implemented remain as future work.

Three of the aforementioned non-run time improvements (e-values, filters and frameshift penalties) require testing that is currently run time prohibitive. Therefore, the first order of business will be to address the areas for potential run time improvement. Yet, even with the potential for improvement, FATHMM is production ready today. I look forward to making it broadly available, and working with collaborators to apply it to a number of datasets containing expected frameshift errors.

BIBLIOGRAPHY

[1] The UniProt Consortium, "UniProt: a worldwide hub of protein knowledge," *Nucleic Acids Research*, vol. 47, no. D1, pp. D506–D515, 11 2018. [Online]. Available: https://doi.org/10.1093/nar/gky1049

[2] S. El-Gebali, J. Mistry, A. Bateman, S. R. Eddy *et al.*, "The Pfam protein families database in 2019," *Nucleic Acids Research*, vol. 47, no. D1, pp. D427–D432, 2019.

[3] S. R. Eddy, "Where did the BLOSUM62 alignment score matrix come from?" *Nature Biotechnology*, vol. 22, no. 8, pp. 1035–1036, 2004.

[4] "BLOSUM62." [Online]. Available: https://www.ncbi.nlm.nih.gov/Class/FieldGuide/BLOSUM62.txt

[5] "The statistics of sequence similarity scores." [Online]. Available: https://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html

[6] S. Henikoff and J. G. Henikoff, "Amino acid substitution matrices from protein blocks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 89, no. 22, pp. 10 915–10 919, 1992.

[7] M.O.Dayhoff, R.M.Schwartz, and B. C. Orcutt, "A Model of Evolutionary Change," *Atlas of protein sequence and structure*, pp. 345–352, 1978.

[8] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.

[9] S. R. Eddy, "Accelerated Profile HMM Searches," *PLoS Computational Biology*, vol. 7, no. 10, 2011.

[10] K. Sjölander, K. Karplus, M. Brown, R. Hughey, A. Krogh, I. S. Mian, and D. Haussler, "Dirichlet mixtures: a method for improved detection of weak but significant protein sequence homology," *Bioinformatics*, vol. 12, no. 4, pp. 327–345, 1996.

[11] S. R. Eddy, "Profile hidden Markov models," *Bioinformatics*, pp. 755–763, 1998.

[12] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis. Probabilistic Models of Proteins and Nucleic Acids.* Cambridge, England: Cambridge University Press, 1999.

[13] S. R. Eddy, "A probabilistic model of local sequence alignment that simplifies statistical significance estimation," *PLoS Computational Biology*, vol. 4, no. 5, 2008.

[14] S. Eddy, "What is dynamic programming?" *Nature Biotechnology*, vol. 22, pp. 909–910, 2004.

[15] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[16] O. R. P. Bininda-Emonds, "TransAlign: Using amino acids to facilitate the multiple alignment of protein-coding DNA sequences," *BMC Bioinformatics*, vol. 6, pp. 1–6, 2005.

[17] R. Wernersson and A. G. Pedersen, "RevTrans : multiple alignment of coding DNA from aligned amino acid sequences," *Nucleic Acids Research*, vol. 31, no. 13, pp. 3537–3539, 2003.

[18] J. C. Dohm, P. Peters, N. Stralis-Pavese, and H. Himmelbauer, "Benchmarking of long-read correction methods," *NAR Genomics and Bioinformatics*, vol. 2, no. 2, pp. 1–12, 2020.

[19] W. R. Pearson, T. Wood, Z. Zhang, and W. Miller, "Comparison of DNA Sequences with Protein Sequences," *Genomics*, vol. 36, no. 46, pp. 24–36, 1997.

[20] S. L. Sheetlin, Y. Park, M. C. Frith, and J. L. Spouge, "Sequence analysis Frameshift alignment : statistics and post-genomic applications," *Bioinformatics*, vol. 30, no. 24, pp. 3575–3582, 2014.

[21] E. Birney, M. Clamp, and R. Durbin, "Genewise and genomewise," *Genome research*, vol. 14, no. 5, pp. 988–995, 2004.

[22] T. J. Wheeler and S. R. Eddy, "nhmmer: Dna homology search with profile hmms," *Bioinformatics*, vol. 29, no. 19, pp. 2487–2489, 2013.

[23] C. Do, M. Mahabhashyam, M. Drudno, and S. Batzoglou, "Probcons: Probabilistic consistency-based multiple sequence alignment," *Genome Research*, vol. 25, no. 2, pp. 330–340, 2005.

[24] M. A. Campbell, P. Łukasik, C. Simon, and J. P. Mccutcheon, "Idiosyncratic genome degradation in a bacterial endosymbiont of periodical cicadas," *Current Biology*, vol. 27, no. 22, 2017.

[25] J. P. Mccutcheon and N. A. Moran, "Extreme genome reduction in symbiotic bacteria," *Nature Reviews Microbiology*, vol. 10, no. 1, p. 13–26, 2011.

[26] D. H. Rich, "Spare forward-backward alignment for sensitive database search with small memory and time requirements," Master's thesis, The University of Montana, 2021.

[27] D. Liu and M. Steinegger, "Block aligner: fast and flexible pairwise sequence alignment with simd-accelerated adaptive blocks," *bioRxiv*, 2021.

[28] M. C. Frith, "How sequence alignment scores correspond to probability models," *Bioinformatics*, vol. 36, no. 2, pp. 408–415, 2020.

[29] W. Shands, "Transmark," 2015. [Online]. Available: https://github.com/TravisWheelerLab/transmark